Configuring GateD

V.9.3.2

Contents

Chapter 1
About this Manual

1.1 Overview

This manual explains how to configure GateD on a UNIX system. For information on how to install GateD, see *Installing GateD*. For information on specific commands, see the *GateD Command Reference Guide*. For information on command line information, including options, signals, and various support programs, see *Operating GateD*.

1.2 Audience

This manual is written for system administrators who are configuring GateD to route packets. You will need to understand basic routing concepts and UNIX commands to understand this manual.

1.3 Configuring GateD Sections

In general, chapters of this manual have four sections:

- Overview
- Syntax
- Sample Configurations
- Protocol-specific Issues

1.3.1 Overview

In the Overview section, you can find an explanation of how the protocol (or GateD statement) works. Overview information might include the RFC (or sections of the RFC) that matches a particular GateD protocol, the algorithms used to route packets, and how GateD's implementation of the protocol differs from other routers.

1.3.2 Syntax

In the Syntax section, you can find all the options and subcommands for the GateD statement, typed exactly as you would type them in your config file if you were to use every option.

Keywords and special characters that the parser expects exactly are displayed in **`courier bold`** font. Parameters are shown in *`courier italics`* font. Optional keywords or parameters are shown in square brackets ('[' and ']'). The vertical bar ('|') is used to indicate a choice of parameters. Parentheses ('(' and ')') are used to group keywords and parameters when necessary.

For example, in the syntax description:

```
[ backbone | ( area areanumber ) ]
```

backbone and **area** are keywords. *areanumber* is a variable of **area**. The square brackets indicate that these keywords are optional. The vertical bar indicates that either "**backbone**" or "**area** *areanumber*" can be specified. Because *areanumber* is in italics, you must provide the value for this variable.

If you want more specific information about a command, refer to the *GateD Command Reference Guide*, which organizes commands alphabetically within protocols.

1.3.3 Sample Configurations

In the Sample Configurations section, you can find examples of valid syntax for the protocol or GateD statement. You can type (or copy and paste) these sections into your configuration file.

1.3.4 Protocol-specific Issues

Any other issues specific to the protocol or to its GateD statement can be found in the sections that follow Sample Configurations. In some cases, we have included the defaults of the protocol.

Chapter 2
Overview and Statement Summary

2.1 What is GateD?

GateD is a modular software program consisting of

- core services
  - Check sum
  - AS Path Regular Expression parsing
  - Multiple RIBs
  - GateD Interactive Interface (GII)
  - Task and Timer functionality
  - Weighted Route Damping
- protocol modules supporting multiple routing protocols
  - RIP versions 1 and 2
  - OSPF
  - IS-IS
  - BGP
  - MP-BGP
  - DVMRP
  - IGMP
  - PIM-SM
  - MSDP
  - IS-IS for v6
  - RIPng

GateD was first used to interconnect the NSFNet and emerging regional networks and to implement policy-based dynamic routing. GateD provides complete policy control for your layer 3 IP routing, which allows a network administrator to control import and export of routing information by

- individual protocol
- source and destination autonomous system
- origin attribute
- source and destination interface
- previous and next hop router
- specific destination address

The network administrator can specify a preference level for each combination of routing information that is imported by using a flexible masking capability. Once the preference levels are assigned, GateD makes a decision about which route to use independent of the protocols involved.

GateD relies on the underlying operating system to provide some facilities, such as forwarding and layer 2 management. It was designed with porting in mind and has been ported to most popular operating systems.

2.2 How to Configure GateD

GateD reads its configuration from a file (`gated.conf`) which consists of a sequence of statements, each terminated by a semi-colon (";"). Statements are composed of commands (and sometimes variables) separated by white space, which can be any combination of blanks, tabs, and newlines. This structure makes it easy to identify parts of the configuration that are associated with each other and with specific protocols. Comments can be specified in either of two forms. One form begins with a pound sign ("#") and runs to the end of the line. The other form, C style, starts with "/*" and continues until it reaches "*/ ."

2.3 Statement Grouping

The configuration statements and the order in which these statements appear divide a `gated.conf` file into six groups:

1. options group (See "Chapter 6 Options Statements" on page 21 for more information.)
2. interface group (See "Chapter 7 Interface Statement" on page 23 for more information.)
3. definition group (See "Chapter 8 Definition Statements" on page 29 for more information.)
4. protocol group (See Chapters 11 through 27 for unicast, multicast, and IPv6 protocol statements.
5. static group (See "Chapter 19 Static Routes" on page 101 for more information.)
6. control and aggregate group (See "Chapter 33 Route Aggregation and Generation" on page 155 for more information.)

See "Table 1: Summary of GateD Configuration Statements" on page 7 for more information about these groupings.

If you enter a grouping out of order, an error occurs when the configuration file is parsed.

Two other types of statements do not fit into these categories: directive statements and trace statements. These statements provide instructions to the parser and control tracing. They do not relate to the configuration of any protocol, and they can occur anywhere in the `gated.conf` file. (See "Chapter 5 Directive Statements" on page 19 for more information about directive statements and "Chapter 4 Trace Statements" on page 15 for more information about trace statements.)

2.4 Address and Prefix Formats

GateD allows configuration of both IPv4 and IPv6 address types. Normally GateD can recognize which type of address is being configured in a particular instance by the format of the address.

IPv4 addresses are 32 bits long. The formats of Ipv4 addresses recognized by GateD are:

d

d.d

d.d.d

d.d.d.d

where *d* represents a number in the range 0-255 inclusive. Each *d* specifies 8 bits of the address. If fewer than four *d* values are provided then the values provided specify the high order values of the address. For example, **192.168** is equivalent to **192.168.0.0**.

For IPv6 addresses, the forms recognized are those specified in RFC 237. IPv6 addresses are 128 bits long and can be specified in one of three forms. In the preferred form, the address is specified as 8 hexadecimal values separated by '**:**'s. Each hexadecimal value specifies 16 bits. The hexadecimal digits a-f can be specified in either upper or lower case. An example of the preferred form is:

3FFd:201:1:7fff:0:0:0:a

The compressed form allows consecutive 0 bits in an IPv6 address to be specified as '**::**'. The **::** may only appear once in a given address. Examples of the compressed form are:

:: (equivalent to **0:0:0:0:0:0:0:0**)

3ffd:201:1:7fff::a (equivalent to the preferred address example above)

::1 (equivalent to **0:0:0:0:0:0:0:1**, the IPv6 loopback address).

The third form provides a way to specify IPv4 addresses to be embedded in an IPv6 address. In this case, the last 32 bits of the address are specified in dotted-quad form (all four *d* values are required). Examples of this address form are:

::192.168.0.0

::ffff:64.32.1.0

In the case where an IPv6 Link-Local address is being configured, GateD allows the interface associated with the Link-Local address to be specified as follows:

fe80::%*interface_name*

where *interface_name* is the name of a physical interface on the machine on which GateD is running. Gated inserts the interface index associated with *interface_name* in the Link-Local address. This form is only valid for Link-Local addresses.

In many cases IPv4 and IPv6 addresses are combined with masks to configure prefixes. There are two methods for specifying the mask: It can be specified as an IPv4 or IPv6 address proceeded by the **mask** keyword; or it can be specified as a length proceeded by the **masklen** keyword or, more conventionally, by a '**/**'. In the "**mask**" case, the address type of the mask must match the address type. Currently only contiguous bit masks are allowed in GateD. Any non-zero address bits in positions that are covered by the specified mask cause a parse error. Example prefix specifications are:

10/8

10.0.0.0 mask 255.0.0.0 (equivalent to 10/8)

10 masklen 8 (equivalent to 10/8)

3ffd::/16

3ffd:: mask ffff:: (equivalent to 3ffd::/16)

3ffd:: masklen 16 (equivalent to 3ffd::/16)

::/0 (IPv6 default address)

0/0 (IPv4 default address)

192.168.1/16 (invalid because the .1 is not covered by the mask)

When configuring route filters (See “Chapter 28 Route Filtering” on page 129) the keywords **default** and **all** are used to indicate an address that matches only the default address and the address that matches any address, respectively. In contexts where either an IPv4 or IPv6 address filter could be specified it is sometimes necessary to proceed either of these keywords with an **inet** or **inet6** to indicate the desired address family. In such contexts, specifying **all** or **default** by itself is taken to mean both address families. For example, within a BGP import or export statement (when MPBGP is enabled)

```
{ all ; } ;
```

Is equivalent to

```
{inet all ; inet6 all ; } ;
```

Note that **ipv4** and **inet4** are synonyms for **inet** and **ipv6** is a synonym for **inet6**.

Although its use usually isn't recommended, GateD provides a feature where host names can be used to configure certain addresses. The syntax for this type of configuration is

```
host [ inet | inet6 ] host_name
```

The optional **inet** or **inet6** keyword indicates the type of address desired as a result of the name resolution. If neither is specified, the type of address sought is determined by context.

The **host** keyword can also proceed an IPv4 or IPv6 address instead of a *host_name*. In this case the mask associated with the prefix will be the host mask for the address family of the specified address. For example

```
host 1.2.3.4
```

is equivalent to

```
1.2.3.4/32
```

In contexts where an IPv4 prefix is allowed, GateD currently allows just an IPv4 address to be specified. It then uses the "natural mask" as the mask for the prefix. While this "feature" is still provided by GateD, its use is strongly discouraged.

2.5 Route Preference and Route Selection

Preference is the value GateD uses to order preference of routes from one protocol or peer over another. Preference can be set in the GateD configuration file in several different configuration statements. Preference can be set based on one network interface over another, from one protocol over another, or from one remote gateway over another. Flexibility of preference configuration varies with the type of protocol being configured. (See “Chapter 3 Preferences and Route Selection” on page 11 for more information about Route Preference.)

2.6 Statement Summary

Table 1 lists each GateD configuration statement by name, indicates the chapter in which the statement is described, identifies the statement group, provides a short synopsis of the

statement's function, and lists the type of protocol. More detailed definitions and descriptions of each of the eight groups of GateD statements follow.

These statements must appear in statement group order in the configuration file. (For example, if definition and protocol statements are both to be included, definition statements must precede the protocol statements.)

Table 1: Summary of GateD Configuration Statements

Statement Name	Chapter in which the Statement is Described	Statement Group	Statement Function	Type of Protocol
traceoptions	Chapter 4	Trace	Global tracing parameters	n/a
%directory	Chapter 5	Directive	Sets the directory for include files	n/a
%include	Chapter 5	Directive	Includes a file in `gated.conf`	n/a
options	Chapter 6	Option	Sets GateD options	n/a
interfaces	Chapter 7	Interface	Defines GateD interfaces	n/a
autonomous-system	Chapter 8	Definition	Sets the AS number for this router	n/a
routerid	Chapter 8	Definition	Sets the router ID for BGP and OSPF	n/a
martians	Chapter 8	Definition	Defines invalid destination addresses	n/a
rip	Chapter 11	Protocol	Configures RIP protocol	Unicast
ospf	Chapter 12	Protocol	Configure OSPF protocol	Unicast
isis	Chapter 13	Protocol	Configures IS-IS protocol	Unicast
bgp	Chapter 14	Protocol	Configures BGP protocol	Unicast

Table 1: Summary of GateD Configuration Statements

Statement Name	Chapter in which the Statement is Described	Statement Group	Statement Function	Type of Protocol
routerdiscovery	Chapter 15	Protocol	Configures the Router Discovery protocol	Pseudo-protocol
icmp	Chapter 16	Protocol	Configures the processing of general ICMP packets	Pseudo-protocol
redirect	Chapter 17	Protocol	Configures the processing of ICMP redirects	Pseudo-protocol
kernel	Chapter 18	Protocol	Configures kernel interaction options	Pseudo-protocol
dvmrp	Chapter 20	Protocol	Configures DVMRP protocol	Multicast
pim	Chapter 21	Protocol	Configures PIM protocol	Multicast
msdp	Chapter 23	Protocol	Configures MSDP protocol	Multicast
igmp	Chapter 24	Protocol	Enables IGMP	Multicast
multicast	Chapter 25	Protocol	Sets interface-specific multicast options	n/a
ripng	Chapter 27	Protocol	Configures RIP next generation protocol	Unicast (v6)
static	Chapter 19	Static	Configures static routes	n/a
mstatic	Chapter 26	Static	Configures static group joins	n/a
import	Chapter 31	Control	Sets policy for the routes installed in the RIBs	n/a

Table 1: Summary of GateD Configuration Statements

Statement Name	Chapter in which the Statement is Described	Statement Group	Statement Function	Type of Protocol
export	Chapter 32	Control	Sets the policy for the routes to export from one protocol to another	n/a
aggregate	Chapter 33	Control	Sets the aggregation policy	n/a
generate	Chapter 33	Control	Sets conditions for generating a default route	n/a

Chapter 3
Preferences and Route Selection

3.1 Preferences Overview

Preference is the value that GateD uses to select one route over another when more than one route to the same destination is learned from different protocols or peers. Preference can be set in the GateD configuration files in several different configuration statements. Preference can be set based on one network interface over another, one protocol over another, or one remote gateway over another. Preference cannot be used to control the selection of routes within an interior gateway protocol. This control is accomplished automatically by the protocol based on metric. Preference can be used to select routes from the same exterior gateway protocol (such as BGP) learned from different peers or autonomous systems. Each route has only one configurable preference value associated with it, even though preference can be set at many places in the configuration file. Simply, the last or most specific preference value set for a route is the value used.

The **`preference`** value is an arbitrarily assigned value used to determine the order of routes to the same destination in a single routing database. The active route is chosen by the lowest **`preference`** value. Some protocols implement a second preference (**`preference2`**), sometimes referred to as a tie-breaker. BGP and OSPF protocols use **`preference2`**. **`preference2`** is for internal use only and is not configurable. Its value is used only when comparing routes with equal values of preference.

3.2 Assigning Preferences

A default preference is assigned to each source from which GateD receives routes. Preference values range from 0 to 255, with the lowest number indicating the most preferred route.

The following table summarizes the default preference values for routes learned in various ways. The table lists the statements (some of which are clauses within statements) that set preference and shows the types of routes to which each statement applies. The table lists the preference precedence between protocols and the default preference for each type of route. The more narrow the scope of the statement, the higher the precedence its preference value is given, but the smaller the set of routes it affects.

Table 2: Preference Selection Precedence

Preference of	Defined by Statement	Default
Direct connected networks	**interface**	0
Routes to interface aliases		1
OSPF routes	**ospf**	10
IS-IS level 1 routes	**isis level 1**	15
IS-IS level 2 routes	**isis level 2**	18
Redirects	**redirect**	30
Routes learned via route socket	**kernel**	40
Routes installed via SNMP		50
Routes learned via router discovery	**router-discovery**	55
Static routes from config	**static**	60
RIP routes	**rip** or **ripng**	100
Point-to-point interface		110
Routes to interfaces that are down	**interface**	120
Aggregate/generate routes	**aggregate/generate**	130
OSPF AS external routes	**ospf**	150
BGP routes	**bgp**	170
Routes in kernel at startup		254

3.3 Sample Preference Configurations

```
interfaces {
    interface 138.66.12.2 preference 10 ;
} ;
rip yes {
    preference 90 ;
} ;
import proto rip gateway 138.66.12.1 preference 75 ;
```

In these statements, the preference applicable to routes learned via RIP from gateway 138.66.12.1 is 75. The last preference applicable to routes learned via RIP from gateway 138.66.12.1 is defined in the **accept** statement. The preference applicable to other RIP routes is found in the **rip** statement. The preference set on the **interface** statement applies only to the route to that interface.

Chapter 4
Trace Statements

4.1 Trace Overview

Trace statements control tracing options. GateD's tracing options may be configured at many levels. Tracing options include the file specifications, control options, and global and protocol-specific tracing options. Unless overridden, tracing options from the next higher level are inherited by lower levels. For example, BGP peer tracing options are inherited from BGP group tracing options, which are inherited from global BGP tracing options, which are inherited from global GateD tracing options. At each level, additional tracing specifications override the inherited options.

When more than one trace options line is used in a section, the "last" trace options line to be parsed by GateD is the one that takes effect. In the case of global tracing, any trace files specified in any trace options line will be created, but tracing will cease for that file when the next trace options line is parsed.

4.2 Trace Syntax

```
traceoptions [ trace_file [ replace ]
     [ size tracesize [ k | m ] files tracefiles ] ]
     [ nostamp ][ trace_global_options ]
     [ except trace_global_options ] ;
traceoptions none ;
```

More detailed descriptions of these commands can be found on page 3 of the *Command Reference Guide*.

This sequence of options is used to specify the name of the trace file (*trace_file*) or files and parameters about these files. Trace files can be specified as a global parameter for all of GateD, for a protocol instance, or for a peer or peers within a protocol.

4.3 Global, Protocol, and Packet Tracing

GateD uses three types of trace options: those that affect only global operations, those that have potential significance to protocols, and those that affect packets.

4.3.1 Global Significance Only

The *trace_options* that have only global significance are:

parse
parse specifies to trace the lexical analyzer and parser. **parse** is used mostly by GateD developers for debugging configuration parsing processing.

adv
adv specifies to trace the allocation of and freeing of policy blocks. **adv** is used mostly by GateD developers for debugging the use of adv-entry structures in parsing.

symbols
symbols specifies to trace symbols read from the kernel at startup. The only useful way to specify this level of tracing is via the `-t` option on the command line, because the symbols are read from the kernel before parsing the configuration file.

iflist
iflist specifies to trace the reading of the kernel interface list. To trace the initial read of the interface information, specify **iflist** with the `-t` option on the command line, because the first interface scan is done before reading the configuration file.

4.3.2 Protocol Significance

The *trace_options* that have potential significance to protocols are:

all
all specifies to turn on **detail**, **packets**, and all of the following:

general
general specifies to trace both **normal** and **route**.

normal
normal specifies to trace normal protocol occurrences. Abnormal protocol occurrences are always traced.

route
route specifies to trace routing table changes for routes installed by this protocol or peer.

state
state specifies to trace state machine transitions in the protocols.

policy
policy specifies to trace application of protocol- and user-specified policy to routes being imported and exported.

task
task specifies to trace system interface and processing associated with this protocol or peer.

timer
timer specifies to trace timer usage by this protocol or peer.

none
none disables all tracing for this protocol or peer.

Not all of the above options apply to all of the protocols. In some cases, their use does not make sense (for instance, RIP does not have a state machine), and in some cases, the requested tracing has not been implemented.

When protocols inherit their tracing options from the global tracing options, tracing options that don't make sense (such as **parse**, **adv**, and **packet** tracing options) are masked out.

Global tracing statements have an immediate effect, especially parsing options that affect the parsing of the configuration file. Tracing values inherited by protocols are initially inherited from the global options that are currently in effect as the protocol configuration entries are parsed, unless they are overridden by more specific options. After the configuration file is read, protocol tracing options that were not explicitly specified are inherited from the global options in effect at the end of the configuration file.

4.3.3 Packet Tracing

These options apply to the protocol-specific **traceoptions** statements, but are supplied here for reference.

[detail] [send | receive] packets

Tracing of packets is very flexible. For any given protocol, there are one or more options for tracing packets. Protocol-specific tracing options are described in the *GateD Command Reference Guide* under the protocols' trace commands. All protocols allow use of the **packets** keyword for tracing all packets sent and received by the protocol. Most protocols have other options for limiting tracing to a useful subset of packet types. These tracing options can be further controlled with the following modifiers:

detail

Normally, packets are traced in a terse form of one or two lines. When **detail** is specified, a more verbose format provides further detail on the contents of the packet. If a protocol allows for several different types of packet tracing, modifiers may be applied to each individual type. However, be aware that within one tracing specification, the trace options are "orred" together, so specifying **detail packets** will turn on full tracing for all packets.

send or **recv**

send or **recv** limit the tracing to packets sent or received, respectively. If neither is specified, both sent and received packets will be traced. Only one of these keywords can be specified in any given instance.

detail, **send**, and **recv** are all optional. If **detail** is specified, it must immediately precede any **send** or **recv** specification. If **detail** and/or **send** or **recv** is specified, at least one packet type trace option must immediately follow.

Chapter 5
Directive Statements

Directive statements provide direction to the GateD configuration language parser about included files and the directories in which these files reside. Directive statements are immediately acted upon by the parser. Other statements terminate with a semicolon (";"), but directive statements terminate with a new line. The two directive statements are:

%directory *"directory"*

%directory defines the directory in which the included files are stored. When **%directory** is used, GateD looks in the directory identified by the path name for any included files that do not have a fully qualified filename (for example, do not begin with "/"). **%directory** does not actually change the current directory; it just specifies the prefix applied to the included file names. Refer to "%directory" on page 7 in the *GateD Command Reference Guide* for additional information.

%include *"filename"*

%include identifies an include file. The content of the file is included in the `gated.conf` file at the point in the `gated.conf` file where the **%include** directive is encountered. If the filename is not fully qualified, (for example, it does not begin with "/"), the file is considered to be relative to the directory defined in the last **%directory** directive. The **%include** directive statement causes the specified file to be parsed completely before resuming with this file. Nesting of up to 10 levels is supported. The maximum nesting level may be increased by changing the definition of **FI_MAX** in *parse.h*. Refer to "%include" on page 8 in the *GateD Command Reference Guide* for additional information.

In a complex environment, segmenting a large configuration into smaller, more easily understood segments might be helpful. One of the great advantages of GateD, however, is that it combines the configuration of several different routing protocols into a single file. Segmenting a small file unnecessarily complicates routing configurations.

Chapter 6
Options Statements

6.1 Options Overview

Options statements allow specification of global options. If used, **options** must appear before any other type of configuration statement in the `gated.conf` file.

6.2 Options Syntax

The options statements syntax is:

```
options
    [ nosend ]
    [ noresolv ]
    [ syslog [ upto ] log_level ]
    [ mark time ]
    ;
```

nosend - causes GateD to never transmit packets. This is useful for testing and debugging.

noresolv - Do not try to resolve host names to IP addresses.

syslog - sets what GateD logs via syslog. Optionally, **upto** can be specified to indicate that everything up to the specified log level is logged. For more information, consult the setlogmask(3) man page.

mark - forces GateD to write a line in the log file every *time* seconds. The line is simply **mark** and the current time. This provides a way to confirm that GateD is still running. Note that GateD does not write the mark line by default.

More detailed descriptions of these commands can be found on page 9 of the *Command Reference Guide*.

Chapter 7
Interface Statement

7.1 Overview

An interface statement connects a router or host to a layer 1 subsystem.

7.2 Interface Syntax

GateD's interface syntax provides support for aliases and tunnels.

```
interfaces {
    [ options
        [ strictinterfaces ]
        [ scaninterval time ]
        [ aliases-nexthop ( primary | lowestip ) ] ; ]
    [ interface interface_list
        [ preference interfacepreference ]
        [ down preference downpreference ]
        [ enable ]
        [ disable ]
        [ passive ]
        [ simplex ]
        [ reject ]
        [ blackhole ]
        [ AS autonomoussystem ]
        [ alias primary address mask mask ]
        [ aliases-nexthop ( primary | lowestip ) ] ; ]
    [ define ( subnet | p2p ) local address
        [ broadcast address ]
        [ remote address ]
        [ tunnel encapsulation_protocol ]
        [ netmask mask ]
        [ multicast | nomulticast ]
        [ unicast | nounicast ] ; ]
```

```
};
```

More detailed descriptions of these commands can be found on page 15 of the *Command Reference Guide*.

7.3 Default Configuration

The default configuration for the **interface** clause is:

```
interfaces {
    options scaninterval 60 ;
    aliases-nexthop primary ;
    interface all down-preference 120 ;
    interface all preference 0 ;
    interface all enable;
} ;
```

Notes:

- On systems without a routing socket, the scaninterval is reduced to 15 seconds to allow GateD to notice changes more quickly.
- On systems with a routing socket, a scaninterval of zero disables periodic interface scans.
- Systems that utilize a routing socket that do not prevent loss of data on the socket may result in a FIB that is inconsistent with GateD's routing table. A scaninterval of zero is highly discouraged on systems with a "lossy" routing socket.

7.4 Sample Interface Configurations

```
interfaces {
    define p2p local 198.108.60.89 remote
        141.213.10.41 multicast nounicast
        tunnel ipip;
    define subnet local 192.168.12.114 netmask
        255.255.255.0;
    define subnet local 192.168.13.129 netmask
        255.255.255.252
        broadcast 192.168.13.131;
    define p2p local 192.168.13.114 remote
        192.168.13.116;
};
```

The **define** statement allows the user to configure an interface that may not exist at the time of setup. The interface may then be used in protocol configurations, with the configuration becoming active when the interface appears and is up.

The first **define** configures a multicast-only IP-in-IP tunnel usable by routing protocols for the multicast RIB. (See "Chapter 9 Multiple Routing Information Bases (RIBs)" on page 33 for more information about multicast RIBs.) Note that the keywords **multicast** and **nounicast** here are redundant with the defaults for **tunnel ipip**. In fact, the standard multicast kernel cannot support any other combination.

The second **define** tells GateD to treat the interface with the local address 192.168.12.114 as a subnet (192.168.12/24), even if it's actually a point-to-point link. (This does, however, require that the actual remote point-to-point address fall within the configured subnet prefix.)

The third **define** shows how a /30 may be implemented in the define statement. The **define** tells GateD to treat the interface with a local address of 192.168.13.129, a netmask of 255.255.255.252, and a broadcast of 192.168.13.131.

The fourth **define** tells GateD to treat the interface with the local address 192.168.13.114 as a point-to-point link to 192.168.13.116, even if it's not actually a point-to-point link. (If it's actually a subnet, this requires that the configured remote point-to-point address fall within the actual subnet prefix.)

7.5 IP Interface Addresses and Routes

The *BSD 4.3* and later networking implementations allow four types of interfaces. Some implementations allow multiple protocol addresses per physical interface. These implementations are mostly based on *BSD 4.3 Reno* or later.

loopback

loopback must have the address of 127.0.0.1. Packets sent to **loopback** are sent back to the originator. This interface is also used as a catch-all interface for implementing other features, such as **reject** and **blackhole** routes. Although a netmask is reported on this interface, it is ignored. Assign an additional address to this interface that is the same as the OSPF or BGP **routerid** to allow routing to a system based on the **routerid** that will work if some interfaces are down. This may require advertising the address in the protocol.

broadcast

broadcast is a multi-access interface capable of a physical level broadcast, such as Ethernet, Token Ring and FDDI. This interface has an associated subnet mask and broadcast address. The interface route to a **broadcast** network will be a route to the complete subnet.

point-to-point

point-to-point is a tunnel to another host, usually on some sort of serial link. This interface has a local address and a remote address. Although it may be possible to specify multiple addresses for a point-to-point interface, there does not seem to be a useful reason for doing so. The remote address must be unique among all the interface addresses on a given router. The local address may be shared among many point-to-point and up to one non-point-to-point interface. **point-to-point** is technically a form of the **routerid** method for addressless links. This technique conserves subnets because none are required when using it.

If a subnet mask is specified on a point-to-point interface, it is only used by RIP version 1 to determine which subnets may be propagated to the router on the other side of this interface. All point-to-point interfaces are, by default, given a host subnet mask in GateD.

non-broadcast multi-access or **nbma**

nbma is multi-access, but not capable of broadcast. An example of this would be `frame relay` and `X.25`. This type of interface has a local address and a subnet mask.

To ensure consistency, GateD installs a route in the kernel's FIB (Forwarding Information Base) for the address of each IP interface that is configured and up.

For point-to-point interfaces, GateD installs some special routes. If the local address on one or more point-to-point interfaces is not shared with a non-point-to-point interface, GateD installs a route to the local address pointing at the **loopback** interface with a preference of **110**. This insures that packets originating on this router destined for this local address are handled locally. OSPF prefers to route packets for the local interface across the point-to-point link where they will be returned by the router on the remote end. This is used to verify operation of the link. Because OSPF installs routes with a preference of **10**, these routes will override the route installed with a preference of **110**.

If the local address of one or more point-to-point interfaces is shared with a non-point-to-point interface, GateD installs a route to the local address with a preference of **0** that will not be installed in the forwarding table. This is to prevent protocols like OSPF from routing packets to this address across a serial interface when this system could be functioning as a host.

When the status of an interface changes, GateD notifies all the protocols, which take the appropriate action. GateD assumes on startup that interfaces that are not marked UP do not exist.

GateD ignores any interfaces that have invalid data for the local, remote or broadcast addresses, or the subnet mask. Invalid data includes zeros in any of these fields. GateD will also ignore any point-to-point interface that has the same local and remote addresses. GateD assumes that the interface is in some sort of loopback test mode.

Interface aging is turned off by default. It can be turned on again via a compile-time option. Refer to the **passive** statement on page 38 of the *GateD Command Reference Guide*.

7.6 Interface Aliases

7.6.1 Aliases Overview

GateD allows the use of aliases on interfaces -- more than one logical interface can exist for each physical interface on the machine. Typically, you create these logical interfaces using the **ifconfig(1)** command. Two options in the **interfaces** command affect the operation of GateD with respect to aliases.

1. **aliases-nh (lowestip | primary)**
2. **interface** *interface-name* **alias primary** *address* **mask** *mask*

The configuration information in the **interfaces** command directly affects the behavior of the protocols when aliases are configured. When used with the **options** command, **aliases-nh** specifies the default behavior. When used with the **interface** command, **aliases-nh** indicates the default for aliases of the physical interface(s) specified.

7.6.2 Using **aliases-nh primary** (default)

When configured with **aliases-nh primary**, which is the default, GateD chooses a primary address on each subnet that is configured on the interface. The primary chosen by GateD is based on the order in which the addresses are read from the kernel. For example, consider a machine with one physical interface, le0, with five logical addresses:

```
le0: flags=1000843 <UP, BROADCAST, RUNNING, MULTICAST, IPv4> mtu 1500
        inet 172.16.0.178 netmask ffff0000 broadcast 172.16.255.255
        inet 172.16.0.179 netmask ffff0000 broadcast 172.16.255.255
        inet 12.1.1.2 netmask ff000000 broadcast 12.255.255.255
        inet 12.1.1.1 netmask ff000000 broadcast 12.255.255.255
        inet 192.168.10.1 netmask ffffff00 broadcast 192.168.10.255
```

In this case, GateD will mark the following interfaces as primary addresses:

- 172.16.0.178 for subnet 172.16.0.0/16
- 12.1.1.2 for subnet 12.0.0.0/8
- 192.168.10.1 for subnet 192.168.10.0/24

The flags for the interface can be seen in the gii **show interfaces** command, in the trace file after an interface scan, or in the GateD dump file. (See "GateD Interactive Interface" in *Operating GateD*, for more information about gii.)

When configured as above, the protocols use the primary address for operation. Attempting to use a logical address that has not been marked as primary will lead to undesired results (to change the primary addresses, see below).

When using a physical interface name in the configuration file, some protocols will attempt to operate on all primary addresses on that interface. Here is an example OSPF statement:

```
ospf yes {
    backbone {
        interface le0 cost 1;
    }
}
```

When configured this way, OSPF will run over the three primary addresses shown above. In the case where there are no neighbors on some of the interfaces, stub links will be announced to these networks. See "Chapter 12 Open Shortest Path First (OSPF)" on page 45 for more information about OSPF.

To mark primary addresses for a subnet in the configuration file, use the **alias primary** option. GateD will allow only one primary address to be configured for each subnet on the interface -- attempting to configure more than one will result in a parse error. Note that, in addition to the interface address, the mask must be specified.

For interface routes, the next hop for a direct subnet will be the primary address.

7.6.3 Using **aliases-nh lowestip**

Versions of GateD prior to 8.0 defaulted to using the lowest IP of an interface for all protocol operations. This feature has been left in place for compatibility. Note that aliases are not really supported with this option; the only valid logical interface is the interface with the numerically lowest IP address.

When configured to use **lowestip**, GateD will install routes to direct nets with a next hop of the lowest IP address for that network configured on the machine. We recommend that operators avoid using this option.

Chapter 8
Definition Statements

8.1 Definition Overview

Definition statements are general configuration statements that relate to all of GateD, or at least to more than one protocol. The four definition statements are **autonomoussystem**, **confed-id**, **routerid**, and **martians**. If used, all of these definition statements must appear before any other type of configuration statement in the `gated.conf` file.

8.2 Autonomous System Syntax

autonomoussystem *autonomous_system* **[loops** *number* **] ;**

autonomoussystem sets the autonomous system (AS) number of this router to be *autonomous_system*. **autonomoussystem** is required if BGP is in use. The AS number is assigned by the Regional Internet Registries (RIRs). When using the BGP confederation option, the AS number is the internal sub-AS number and should be allocated out of the reserved private AS space, 64512-65534. See RFC 3065, "BGP Confederations" and RFC 1930, "Guidelines for Creation, Selection and Registration of an Autonomous System" for details.

loops is only for protocols supporting AS paths. For example, BGP **loops** controls the number of times this autonomous system can appear in an AS path. It defaults to 1. **loops** should not be used in normal operations.

More detailed descriptions of these commands can be found on page 49 of the *Command Reference Guide*.

8.3 Confed ID Syntax

confed-id *confederation_number* **;**

confed-id sets the confederation ID for this router to be *confederation_number*. **confed-id** is required if this router will be using the BGP confederations option. Additionally, *autonomoussystem* must also be defined.

The AS number of the *confederation_number* should be selected out of the reserved private AS space, 64512-65534, as specified in RFC 1930, "Guidelines for Creation, Selection and Registration of an Autonomous System." Non-private ASs, however, can also be selected.

More detailed descriptions of these commands can be found on page 51 of the *Command Reference Guide*.

8.4 Router ID Syntax

```
routerid host ;
```

routerid sets the router identifier for use by the BGP and OSPF protocols. **routerid** must be explicitly configured when using BGP. The default is selected by going through the list of interfaces and using the local address of the most preferred interface. The most preferred interface is selected as follows: the address of a non-point-to-point interface is preferred over the local address of a point-to-point interface, and an address on a loopback interface that is not the loopback address (127.0.0.1) is most preferred.

More detailed descriptions of this command can be found on page 55 of the *Command Reference Guide*.

8.5 Martian Syntax

```
martians {
    host [ inet6 ] host [ allow ] ;
    network [ ( mask mask ) | ( masklen number ) ]
        [ exact | refines | ( between lower and upper ) ]
        [ allow ] ;
    [ inet | inet6 ] default [ allow ] ;
} ;
```

martians allows additions to the list of martian addresses. See the section, “Chapter 28 Route Filtering” on page 129 for more information on specifying ranges. Also, the **allow** parameter may be specified to explicitly allow a subset of a range that was disallowed.

More detailed descriptions of these commands can be found on page 52 of the *Command Reference Guide*.

Martians are networks that are considered illegal to be routed on the internet. RFC 1918 specifies these networks that are part of the private internet space:

- 10.0.0.0 - 10.255.255.255 (10/8 prefix)
- 172.16 - 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

The prefixes are considered unroutable. GateD does not treat these as martian addresses, but the **martian** syntax will allow you to treat private address space as illegal for routing within an autonomous system. RFC 1700 specifies common usage for IP numbers.

The default martians are:

8.5.1 IPv4 Defaults

127/8 (127.0.0.0 netmask 255.0.0.0) - 127.x.x.x is specified by RFC 1700 to loop back addresses. RFC 1700 (page 4, item g) states “these addresses should never appear outside a host.” Address 127.0.0.1 is normally used as a loopback address.

240.0.0.0/4 (240.0.0.0 netmask 240.0.0.0) - 240.x.x.x are the multicast addresses.

8.5.2 IPv6 Defaults

::1/128 - IPv6 loopback address

fe80::/10 - IPv6 link local addresses

ff00::/8 - IPv6 multicast addresses

::0000: 127.0.0.0/104 and ::ffff: 127.0.0.0/104 - IPv6 embedded IPv4 loopback addresses

::0000: 240.0.0.0/100 and ::ffff 240.0.0.0/100 - IPv6 embedded IPv4 multicast addresses

8.6 Martian Examples

This example **martian** statement prevents routes to 10.0.0.26 from being accepted. It also causes routes to 3ffd:ffff:ffff:1::/64 and any more specific subnets or hosts of this route to not be accepted. It also prevents routes to 0.0.0.0/0 and ::/0 from being accepted.

```
martians {
    10.0.0.26 ;
    3ffd:ffff:ffff:1::/64 ;
    default ;
} ;
```

Chapter 9
Multiple Routing Information Bases (RIBs)

9.1 Multiple RIBs Overview

GateD keeps multiple Routing Information Bases (RIBs) with active routes. Currently, two RIBs per address family are available: unicast and multicast. Routes in the unicast RIB get installed in the kernel forwarding information base (FIB) (because the UNIX kernel supports only unicast routes in the FIB). The multicast RIB is used by multicast routing protocols to construct multicast trees. Multicast routes are then installed in the kernel's multicast forwarding cache. Each route may be active in one or more RIBs simultaneously.

9.2 Direct (Interface) Routes

The direct route(s) for each multicast-capable interface apply to (are eligible to become active in) the multicast RIB. The direct route(s) for each unicast-capable interface apply to the unicast RIB. No additional configuration is needed to achieve this.

9.3 Static Routes

Static routes can be tagged with one or more RIB names. By default, a static route applies only to the unicast RIB. (See "Chapter 19 Static Routes" on page 101 for more information about static routes.)

Example:

```
static {
   10.0.0.0 masklen 24 interface le1;
   10.0.1.0 masklen 24 interface le1 unicast;
   10.0.2.0 masklen 24 interface le1 multicast;
   10.0.3.0 masklen 24 interface le1 unicast multicast;
};
```

The first two static routes apply only to the unicast RIB. The third applies only to the multicast RIB, and the last applies to both.

9.4 Aggregate Routes

RIBs need not be specified for aggregate routes. (See "Chapter 33 Route Aggregation and Generation" on page 155 for more information about aggregate routes.) By default, an aggregate applies to all RIBs to which any contributing route applies. For example, an aggregate applies to the unicast RIB if and only if any contributing route applies to the unicast RIB.

Example:

```
aggregate 10.0.0.0 masklen 8 {
    proto static {
        10.0.0.0 masklen 8 refines;
    };
};
```

If any static route in the unicast RIB matches the route filter (which three of the four static routes in the previous example do), the aggregate will exist in the unicast RIB. Likewise, for the multicast RIB.

RIB limits may, however, be specified. By default, the limit is all RIBs (for example, all RIBs to which any contributing route applies). This default can be overridden with a more specific limit, as in the example below.

```
aggregate 10.0.0.0 masklen 8 unicast {
   proto static {
       10.0.0.0 masklen 8 refines;
    };
};
```

The above aggregate applies only to the unicast RIB and only if a contributing route is in the unicast RIB. Contributing routes in other RIBs are ignored.

9.5 Importing Routes

Normally, routes from unicast routing protocols are imported only into the unicast RIB. (See “Chapter 31 Route Importation” on page 137 for more information.) Routes from multicast routing protocols (for example, DVMRP) are imported only into the multicast RIB. However, some multicast routing protocols do not maintain their own routing table. Instead, they rely on the unicast routing protocol. To support these protocols, unicast routes must be imported into the multicast RIB. If this is not done, only interface routes and properly configured static or aggregate routes will be available to these multicast protocols.

Because BGP is able to tag routes as to which RIBs they apply, no additional configuration is required for BGP routes. The RIP and Redirect protocols, however, do not do this. Hence, GateD must be configured to import RIP or Redirect routes into the multicast RIB.

One or more RIB names can be specified as follows (where **multicast** and **unicast** appear below):

```
import proto ( rip | redirect )
    [ ( interface interface_list ) | ( gateway gateway_list ) ]
        [ preference preference ] [ multicast ][ unicast ] {
            {route_filter [ restrict | ( preference preference ) ]
            [ multicast ] [ unicast ];
};
```

If no RIBs are specified, the unicast RIB (only) is assumed.

Example 1

Example 1 keeps the normal behavior of allowing all RIP routes in the unicast RIB, but also imports all routes falling under 198/8 into the multicast RIB.

```
import proto rip  {
    0.0.0.0 masklen 0 refines;
    198.0.0.0 masklen 8 refines multicast unicast;
};
```

Example 2

Example 2 imports all RIP routes (except default) into the multicast RIB (as well as the usual unicast RIB).

```
import proto rip  {
    0.0.0.0 masklen 0 refines multicast unicast;
};
```

To import OSPF routes into the multicast RIB, you currently must import all OSPF routes as follows:

```
ospf yes {
    defaults {
      ribs unicast multicast;
      ...
    };
    ...
};
```

You cannot import OSPF routes into only the multicast RIB. Attempting to do so will be flagged as a configuration error.

9.6 gii

In gii, the **show ip walkup** and **show ip walkdown** commands have been expanded to allow a RIB name as an additional optional argument. If no RIB is specified, the output covers all RIBs combined. Also, another column has been added to their output to show to which RIBs a route applies (“u” for unicast, “m” for multicast).

Example 1

```
GateD> sh ip walkdown 10.0.0.0/8
100 um Agg   10/8  ---     IGP (Id 1)
100 u  Sta   10/24 192.168.10.89 IGP (Id 1)
100 u  Sta   10.0.1/24 192.168.10.89 IGP (Id 1)
100 m  Sta   10.0.2/24 192.168.10.89 IGP (Id 1)
100 um Sta   10.0.3/24 192.168.10.89 IGP (Id 1)
GateD> sh ip walkdown 10.0.0.0/8 unicast
100 u  Agg   10/8  ---  IGP (Id 1)
100 u  Sta   10/24 192.168.10.89  IGP (Id 1)
100 u  Sta   10.0.1/24 192.168.10.89  IGP (Id 1)
100 u  Sta   10.0.3/24 192.168.10.89  IGP (Id 1)
```

```
GateD> sh ip walkdown 10.0.0.0/8 m
100 m  Agg   10/8 ---       IGP (Id 1)
100 m  Sta   10.0.2/24 192.168.10.89  IGP (Id 1)
100 m  Sta   10.0.3/24 192.168.10.89  IGP (Id 1)
GateD>
```

See Chapter 7 of *Operating GateD* for more information about gii, the GateD Interactive Interface.

Chapter 10
Configuration Initialization and Re-initialization

10.1 Overview

When GateD is started or re-initialized (with the HUP signal), it goes through the following series of events.

10.2 Phase 1 - Initialization of Process

At startup or reinit time, GateD attempts to find the state of the kernel routing table and the configuration of the machine's interfaces.

Note: In order to do the former, GateD must be running as root.

The process goes through the following sequence:

10.2.1 Reading the Kernel Routing Table

The reading of the kernel routing table is only done once to find the initial state of the table. After it has been read, GateD listens for changes via the routing socket, kmem, or ioctl interfaces. Which one GateD uses depends on the operating system. (See “Chapter 18 Kernel Interface” on page 95 for more information.)

10.2.2 Reading the Kernel Interface List

At startup and reinit time, as well as periodically during operation, GateD will scan the list of interfaces. It does this using one of several methods, depending on the operating system. Any time the interface list is scanned, the entire list is read and changes are reported to the protocols individually.

10.3 Phase 2 - Initialization of Tasks

Each task has a callback hook associated with initialization, pre-parse initialization, and policy initialization, and interface changes among other things. At startup, each protocol's callback is called in this order:

1. **`var init`** - invokes the protocol’s **`var_init()`** procedure
2. **`parse`** - parses configuration file
3. **`init`** - initializes the protocol with parser information
4. **`reinit`** - invokes the task’s reinit procedure
5. **`if_notify`** - notifies tasks of changes of all physical and logical interfaces
6. **`reinit finalize`** - calls the task’s **`reinit_finalize`** procedures
7. **`new policy`** - configured policy is processed. The task’s **`newpolicy`** procedures are invoked.

Some protocols defer initialization of protocol interface structures until the first Interface Change notification, and at that time, they are added or deleted according to the configuration of the machine and the information in the config file. See "Chapter 9, Interacting with GateD" of *Operating GateD* for more information about how GateD interacts with system administration actions.

10.4 Phase 3 - Re-initialization of Tasks

GateD can be re-initialized while running without disturbing the operation of existing protocol sessions. Reconfiguration can enable instances of protocols that were previously disabled and can disable protocols that are operating at the time of re-initialization. GateD rereads its configuration file upon receiving a SIGHUP signal and processes the changes between its current configuration and that represented in the new instance of the configuration file.

Upon receipt of a SIGHUP signal, GateD calls each running task's cleanup function to give each task an indication that a re-initialization is taking place. A task's cleanup routine basically saves parts of its current configuration state and frees up other parts, such as policy, that can safely be rebuilt without knowing its previous state. Then the seven initialization steps listed above are performed again. As part of re-initialization, all routes are run through both import and export policy again because the policy may have changed. Routes that were imported before the re-initialization may no longer be importable under the new policy, and routes that were unimportable under the old policy may now be importable. The same holds true for the exportation of routes. A change to the importation of routes requires flashing the changes to GateD's route table to all protocols.

Chapter 11
Routing Information Protocol (RIP)

11.1 Overview

One of the most widely used interior gateway protocols is the Routing Information Protocol (RIP). RIP is an implementation of a distance-vector, or Bellman-Ford, algorithm. RIP classifies routers as active and passive (silent). Active routers advertise their routes (reachability information) to others; passive routers listen and update their routes based on advertisements, but do not advertise. Typically, routers run RIP in active mode, while hosts use passive mode.

A router running RIP in active mode sends updates at set intervals. Each update contains paired values, where each pair consists of an IP network address and an integer distance to that network. RIP uses a hop count metric to measure the distance to a destination. In the RIP metric, a router advertises directly connected networks at a metric of 1 by default. Networks that are reachable through one other gateway are 2 hops, etc. Thus, the number of hops or hop count along a path from a given source to a given destination refers to the number of gateways that a datagram would encounter along that path. Using hop counts to calculate shortest paths does not always produce optimal results. For example, a path with a hop count 3 that crosses three Ethernets may be substantially faster than a path with a hop count 2 that crosses two slow-speed serial lines. To compensate for differences in technology, many routers advertise artificially high hop counts for slow links.

RIP dynamically builds on information received through RIP updates. When started up, RIP issues a request for routing information and then listens for responses to the request. If a system configured to supply RIP hears the request, it responds with a response packet based on information in its routing database. The response packet contains destination network addresses and the routing metric for each destination.

When a RIP response packet is received, the routing daemon takes the information and rebuilds the routing database, adding new routes and “better” (lower metric) routes to destinations already listed in the database. RIP also deletes routes from the database if the next router to that destination reports that the route contains more than 15 hops, or if the route is deleted. All routes through a gateway are deleted if no updates are received from that gateway for a specified time period. In general, routing updates are issued every 30 seconds. In many implementations, if a gateway is not heard from for 180 seconds, all routes from that gateway are deleted from the routing database. This 180-second interval also applies to deletion of specific routes.

RIP version 2 (more commonly known as RIP II) adds additional capabilities to RIP. Some of these capabilities are compatible with RIP I and some are not. To avoid supplying information to RIP I routes that could be misinterpreted, RIP II can use only non-compatible fea-

tures when its packets are multicast. On interfaces that are not capable of IP multicast, RIP-I-compatible packets are used that do not contain potentially confusing information.

Some of the most notable RIP II enhancements are:

- Next hop
- Network mask
- Authentication
- RIP tag field

These features in RIP I and II are contrasted in the following paragraphs.

Next hop

With RIP II, a router can advertise a next hop other than itself. Next hop is useful when advertising a static route to a dumb router that does not run RIP, because it avoids having packets that are passed through the dumb router from having to cross a network twice. Because RIP I routers will ignore next hop information in RIP II packets, packets might cross a network twice, which is exactly what happens with RIP I. Next hop information is provided in RIP-I-compatible RIP II packets.

Network mask

RIP I assumes that all subnetworks of a given network are classful (Class A,B,C). RIP I uses this assumption to calculate the network masks for all routes received. This assumption prevents subnets with classless netmasks from being included in RIP packets. RIP II adds the ability to specify the network mask with each network in a packet. Because RIP I routers will ignore the network mask in RIP II packets, their calculation of the network mask will quite possibly be wrong. For this reason, RIP-I-compatible RIP II packets must not contain networks that would be misinterpreted. These networks must be provided only in native RIP II packets that are multicast.

RIP I derives the network mask of received networks and hosts from the network mask of the interface via which the packet was received. If a received network or host is on the same natural network as the interface over which it was received, and that network is subnetted (the specified mask is more or less specific than the natural netmask), the interface's subnet mask is applied to the destination. If bits outside the mask are set, it is assumed to be a host; otherwise, it is assumed to be a subnet. On point-to-point interfaces, the netmask is applied to the remote address. The netmask on these interfaces is ignored if it matches the natural network of the remote address, or is all ones. Unlike previous releases, the zero subnet (a subnetwork that matches the natural network of the interface, but has a more specific, or longer, network mask) is advertised. If this is not desirable, a route filter may be used to reject it.

Authentication

RIP II packets may contain one of two types of authentication strings that may be used to verify the validity of the supplied routing data. Authentication may be used in RIP-I-compatible RIP II packets, but be aware that RIP I routers will ignore these packets (unless **nocheckzero** is selected). The first method is a simple password in which an authentication key of up to 16 characters is included in the packet. If this key does not match what is expected, the packet will be discarded. This method provides very little security because it is possible to learn the authentication key by watching RIP packets.

The second method uses the MD5 algorithm to create a crypto-checksum of a RIP packet and an authentication key of up to 16 characters. The transmitted packet does not contain

the authentication key itself; instead, it contains a crypto-checksum, called the “digest”. The receiving router will perform a calculation using the correct authentication key and discard the packet if the digest does not match. In addition, a sequence number is maintained to prevent the replay of older packets. This method provides a much stronger assurance that routing data originated from a router with a valid authentication key.

Two authentication methods can be specified per interface. Packets are always sent using the primary method, but received packets are checked with both the primary and secondary methods before being discarded. In addition, a separate authentication key is used for non-router queries.

RIP tag field

RIP tags are supported by this implementation.

11.2 RIP Syntax

```
rip ( on | off ) [ {
    broadcast | nobroadcast ;
    ignorehostroutes ;
    expire-time expire_time ;
    update-time update_time ;
    nocheckzero ;
    preference preference ;
    defaultmetric metric ;
    query authentication none ;
    query authentication simple password ;
    query authentication md5 password ;
    query authentication md5 key password id number [ {
        [ start-accept YYYY/MM/DD HH:MM ] ;
        [ stop-accept YYYY/MM/DD HH:MM ] ;
        [ start-generate YYYY/MM/DD HH:MM ] ;
        [ stop-generate YYYY/MM/DD HH:MM ] ;
    } ; ]
    interface interface_list
        [ noripin | ripin ]
        [ noripout | ripout ]
        [ metricin metric ]
        [ metricout metric ]
        [ version 1 | ( version 2 [ multicast | broadcast ] ) ]
        [ secondary ] authentication none ;
        [ secondary ] authentication simple password ;
        [ secondary ] authentication md5 password ;
        [ secondary ] authentication md5 key password id number [ {
            [ start-accept YYYY/MM/DD HH:MM ] ;
            [ stop-accept YYYY/MM/DD HH:MM ] ;
            [ start-generate YYYY/MM/DD HH:MM ] ;
            [ stop-generate YYYY/MM/DD HH:MM ] ;
         } ; ]
    ;
    trustedgateways gateway_list ;
```

```
        sourcegateways gateway_list ;
        traceoptions trace_options ;
    } ] ;
```

More detailed descriptions of these commands can be found on page 57 of the *Command Reference Guide*. See “Chapter 32 Route Exportation” on page 145 for information about exporting and RIP.

11.3 RIP Sample Configurations

11.3.1 RIP version 1 with Broadcast

This configuration broadcasts RIP updates and listens for RIP updates on a single interface. Note that more than one interface must be enabled and IP forwarding must be enabled in order to broadcast RIP updates by default. **broadcast** forces broadcast of RIP updates.

```
rip yes {
        traceoptions all ;
        broadcast ;
        interface fxp0 ripin ripout ;
        interface fxp1 noripin noripout ;
        interface fxp2 noripin noripout ;
};
```

11.3.2 RIP version 2 with Broadcast and Simple Authentication

This configuration broadcasts RIP updates and listens for RIP updates on a single interface. Version 2 and simple authentication are used.

```
rip yes {
        traceoptions all ;
        broadcast ;
        interface fxp0 ripin ripout version 2 authentication simple
"foo" ;
        interface 10.3.25.25 noripin noripout ;
};
```

11.3.3 RIP version 2 with Multicast and Simple Authentication

This configuration enables version 2 multicast on two interfaces. Note that multicast is the default if version 2 is specified.

```
rip yes {
        traceoptions all ;
        interface fxp0 version 2 authentication simple "foo" ;
        interface fxp1 version 2 authentication simple "bar" ;
};
```

11.3.4 RIP version 2 with Broadcast and MD5 Authentication

This configuration enables a single interface for version 2 RIP with an MD5 authentication key. By default, the key has an infinite lifetime.

```
rip yes {
        traceoptions all;
        broadcast ;
        interface fxp0 version 2 authentication md5 key “foo” id 20;
};
```

11.3.5 RIP version 2 with Source and Trusted Gateways

This configuration uses **sourcegateways** and **trustedgateways** to enable GateD to announce RIP to a single gateway and receive RIP from a single gateway.

```
rip yes {
        traceoptions all ;
        nobroadcast ;
        sourcegateways 10.131.10.12 ;
        trustedgateways 10.131.10.12 ;
        interface 10.131.10.16 version 2 authentication simple "foo";
};
```

Chapter 12
Open Shortest Path First (OSPF)

12.1 OSPF Overview

Open Shortest Path First Routing (OSPF) is a shortest path first or link-state protocol. OSPF is an interior gateway protocol that distributes routing information between routers in a single autonomous system (AS). OSPF chooses the least-cost path as the best path. OSPF is suitable for complex networks with a large number of routers because it provides equal-cost multi-path routing, where packets to a single destination can be sent via more than one interface simultaneously.

In a link-state protocol, each router maintains a database describing the entire AS topology, which it builds out of the collected link state advertisements of all routers. Each participating router distributes its local state (i.e., the router's usable interfaces and reachable neighbors) throughout the AS by flooding. Each multi-access network that has at least two attached routers has a designated router and a backup designated router. The designated router floods a link state advertisement for the multi-access network and has other special responsibilities. The designated router concept reduces the number of adjacencies required on a multi-access network.

OSPF allows networks to be grouped into areas. Routing information passed between areas is abstracted, potentially allowing a significant reduction in routing traffic. OSPF uses the following four different types of routes, listed in order of preference: intra-area, inter-area, type 1 Autonomous System External (ASE), and type 2 ASE. Intra-area paths have destinations within the same area. Inter-area paths have destinations in other OSPF areas. Both types of ASE routes are routes to destinations external to OSPF (and usually external to the AS). Routes exported into OSPF ASE as type 1 ASE routes (see "Exporting to OSPF ASE and NSSA" on page 151) are supposed to be from interior gateway protocols (such as RIP) whose external metrics are directly comparable to OSPF metrics. When a routing decision is being made, OSPF will add the internal cost to the AS border router to the external metric. Type 2 ASEs are used for exterior gateway protocols whose metrics are not comparable to OSPF metrics. In this case, only the internal OSPF cost to the AS border router is used in the routing decision.

From the topology database, each router constructs a tree of the shortest paths, with itself as the root. This shortest-path tree gives the route to each destination in the AS. Externally-derived routing information appears on the tree as leaves. The link-state advertisement format distinguishes between information acquired from external sources and information acquired from internal routers, so there is no ambiguity about the source or reliability of routes. Externally-derived routing information (for example, routes learned from BGP) is passed transparently through the AS and is kept separate from OSPF's internally derived

data. Each external route can also be tagged by the advertising router, enabling routers on the borders of the AS to pass additional information between them.

OSPF optionally includes Type of Service (TOS) routing and allows administrators to install multiple routes to a given destination for each type of services, such as low delay or high throughput. A router running OSPF uses the destination address and the type of service to choose the best route to the destination.

OSPF intra- and inter-area routes are always imported into the GateD routing database with a preference of 10. Because it would violate the protocol if an OSPF router did not participate fully in the area's OSPF, it is not possible to override this preference. Although it is possible to give other routes better preference values explicitly, doing so would violate the OSPF protocol and could lead to incorrect routing. Therefore, you cannot specify import or export policy for OSPF; you can only specify export policy for OSPF ASE.

Hardware multicast capabilities are also used where possible to deliver link-status messages. OSPF areas are connected by the backbone area, the area with identifier 0.0.0.0. All areas must be logically contiguous, and the backbone is no exception. To permit maximum flexibility, OSPF allows the configuration of virtual links, which enables the backbone area to appear contiguous despite the physical reality.

Because a separate copy of the link-state algorithm is run for each area, most configuration parameters are defined on a per-area basis. All routers in an area must agree on that area's parameters. Misconfiguration will keep neighbors from forming adjacencies between themselves, and routing information might not flow or could loop.

GateD can run over a variety of physical connections: serial connections, LAN interfaces, ATM, or FDDI. The OSPF configuration supports three different types of connections in the interface clauses:

LAN and Point-to-Point

An example of a LAN interface is an Ethernet or a FDDI interface. A point-to-point interface can be a serial line using Point-to-Point protocol. GateD will use a Multicast IP address on LAN interfaces to reach OSPF routers.

Non-Broadcast Multiple Access

ATM with virtual circuits is an example of a Non-Broadcast Multiple Access medium. Because there is no general multicast in all ATM devices, each router must be listed so that GateD can poll each router. GateD will unicast the packets to the routers in the NBMA network.

Point-to-Multipoint

Point-to-Multipoint connectivity is used when the network does not provide full connectivity to all routers in the network. Just as on the NBMA format, you must provide a list of routers that the GateD deamon will query as OSPF peers.

12.2 OSPF Syntax

```
ospf on | off [ {
    always-update-summary on | off ;
    retransmitinterval global_default_time ;
```

```
transitdelay global_default_time ;
priority global_default_priority ;
hellointerval global_time ;
routerdeadinterval global_default_time ;
pollinterval global_default_time ;
advertise-subnet on | off ;
opaque-capability on | off ;
auth [none | simple auth_key | md5 md5-keyset]  ;
defaults {
    preference defasepref ;
    cost defasecost ;
    tag [ as ] tagvalue ;
    type 1 | 2 ;
    inherit-metric ;
    ribs unicast [ multicast ] ;
    nssa-preference defnssapref ;
    nssa-cost defnssacost ;
    nssa-type 1 | 2 ;
    nssa-inherit-metric ;
};
traceoptions trace_options_ospf ;
rfc1583compatibility on | off ;
area areanumber | backbone {
    nssa [ cost defaultcost type 1 | 2 ] ;
    nssanetworks {
        network mask stubmask [ restrict ] ;
        network masklen number [ restrict ] ;
        host stubhost [ restrict ] ;
    };
    stub [ cost stub_default_cost] ;
    stubhosts {
        host cost cost ;
    };
    stubnetworks {
        network mask stubmask cost cost ;
        network masklen number cost cost ;
        host stubhost cost cost ;
    };
```

```
networks {
    network [ restrict ] ;
    network mask netmask [ restrict ] ;
    network masklen number [ restrict ] ;
    host nethost [ restrict ] ;
} ;
summaryfilters {
    route_filter
} ;
retransmitinterval area_default_time ;
transitdelay area_default_time ;
priority area_default_priority ;
hellointerval area_time ;
routerdeadinterval area_default_time ;
pollinterval area_default_time ;
advertise-subnet on | off ;
auth [none | simple auth_key | md5 md5-keyset]  ;
interface interface_list
    [ cost ifcost ]
    [ { enable | disable ;
    retransmitinterval iftime ;
    transitdelay iftime ;
    priority ifpriority ;
    hellointerval if_time ;
    routerdeadinterval iftime ;
    pollinterval iftime ;
    passive ;
    advertise-subnet on | off ;
    auth [none | simple auth_key | md5 md5-keyset] ;
    } ] ;
interface interface_name | interface_address nonbroadcast
    [ cost ifnbcost ]
    [ { strict-routers on | off ;
    routers {
        gatewaylist [ eligible ] ;
    } ;
    retransmitinterval ifnbtime ;
    transitdelay ifnbtime ;
```

```
            priority ifnbpriority ;
            hellointerval ifnb_time ;
            routerdeadinterval ifnbtime ;
            pollinterval ifnbtime ;
            passive ;
            advertise-subnet on | off ;
            auth [none | simple auth_key | md5 md5-keyset] ;
            } ] ;
        interface interface_name | interface_address point-to-multipoint
            [ cost ptmcost ]
            [ { strict-routers on | off ;
            routers {
                gatewaylist ;
            } ;
            retransmitinterval ptmtime ;
            transitdelay ptmtime ;
            priority ptmpriority ;
            hellointerval ptmtime ;
            routerdeadinterval ptmtime ;
            pollinterval ptmtime ;
            passive ;
            advertise-subnet on | off ;
            auth [none | simple auth_key | md5 md5-keyset]  ;
            } ] ;
```

Backbone only:

```
virtuallink neighborid router_id
        transitarea area [ {
            retransmitinterval vl_time ;
            transitdelay vl_time ;
            priority vl_priority ;
            hellointerval vl_time ;
            routerdeadinterval vl_time ;
            pollinterval vl_time ;
            passive ;
            advertise-subnet on | off ;
            auth [none | simple auth_key | md5 md5-keyset] ;
        } ] ;
    } ;
```

```
} ] ;
```

More detailed descriptions of these commands can be found on page 95 of the *Command Reference Guide*. See “Exporting to OSPF ASE and NSSA” on page 151 for information about exporting and OSPF.

12.3 OSPF Sample Configurations

12.3.1 Example 1, Host Configuration

The simplest configuration for a host user is the following, which will set GateD into the backbone area specified for all interfaces. The GateD OSPF has defaults for a host that will not allow it to become a designated router (DR) for OSPF.

```
ospf yes;
```

The simplest configuration for a host user in an area outside the backbone is:

```
ospf yes {
    area 0.0.0.2; {
        interface all;
    };
};
```

12.3.2 Example 2, Router Configurations

The same simplest configurations can also be used for UNIX system running as a router. The following configuration is for a router in the backbone.

```
ospf yes {
    priority 1;
    backbone {
        interface all;
    };
};
```

The following gives the same configuration as above.

```
ospf yes { priority 1; };
```

The following configuration is for a router in area 0.0.0.2.

```
ospf yes {
    priority 1;
    area 0.0.0.2 {
        interface all;
    };
};
```

The following configuration is for a simple border router.

```
ospf yes {
    priority 1;
    backbone {
        interface fxp0;
    };
    area 0.0.0.1 {
        interface fxp1;
    };
};
```

Use area ranges to reduce the amount of routing information in the OSPF domain. In this example, area 0.0.0.1 is the only area with 192.168.x/24 networks in it. By specifying a network range, only a single LSA is announced to the backbone (and thus to other areas) advertising the larger 192.168/16 route. The following configuration is for a border router with summarizing area range.

```
ospf yes {
    priority 1;
    backbone {
        interface fxp0;
    };
    area 0.0.0.1 {
        networks {
            192.168 masklen 16;
        };
        interface fxp1;
    };
};
```

To reduce the amount of routing information in a single area, make it a stub area. Stub areas do not receive LSAs for external routes being readvertised in OSPF. Normally, you would also originate a default summary route into the area, so that internal routers have a route to networks outside of the area. The following configuration is for a border router attaching to a stub area and injecting a default route.

```
ospf yes {
    priority 1;
    backbone {
        interface fxp0;
    };
```

```
        area 0.0.0.1 {
            stub cost 1;
            interface fxp1;
        };
    };
```

To further reduce the amount of routing information, when using stub areas, you can filter all (or some subset) of the summary (except the generated default). Be sure to specify the **cost 1** part of the stub statement so that a default route is generated for the routers in the stub area. The following configuration is for a border router attaching to stub area, injecting a default route and filtering all summary.

```
    ospf yes {
        priority 1;
        backbone {
            interface fxp0;
        };
        area 0.0.0.1 {
            stub cost 1;
            summary-filters { all; };
            interface fxp1;
        };
    };
```

12.4 Authentication

By definition, all OSPF protocol exchanges are authenticated; however, one method of authentication is none. Authentication can help to guarantee that routing information is imported only from trusted routers. A variety of authentication schemes can be used, but a single scheme must be configured for each network. The use of different schemes enables some interfaces to use much stricter authentication than others. The three authentication schemes available are: none, simple, and MD5.

No Authentication

When no authentication is required, use authentication type **none**. To use authentication type **none**, add the following line to the appropriate OSPF interface statements.

```
    auth none ;
```

Simple Authentication Key

When you want to keep certain routers from exchanging OSPF packets, use the simple form of authentication. The interfaces that the packets are to be sent on still need to be trusted, because the key will be placed in the packets and can be seen by anyone with access to the network. To specify authentication type **simple**, add the following line to your OSPF interface statements:

```
auth simple "auth-key"
```

where *auth-key* is a quoted string of up to eight characters.

MD5 Authentication

When you do not trust other users of your network, use MD5 authentication. The system works by using shared secret keys. Because the keys are used to sign the packets with an MD5 checksum, they cannot be forged or tampered with. Because the keys are not included in the packet, snooping the key is not possible. Users of the network can still snoop the contents of packets, however, because the packets are not encrypted.

GateD's MD5 authentication is compliant with the specification in OSPF RFC 2328. This specification uses the MD5 algorithm and an authentication key of up to 16 characters. RFC 2328 allows multiple MD5 keys per interface. Each key has two associated time ranges.

To specify a single MD5 key on an interface, add the following to the appropriate OSPF interface statements:

```
auth md5 md5-keyset
```

where *md5-keyset* is:

```
key auth-key id id-number [ {
    [start-generate date-time;]
    [stop-generate date-time;]
    [start-accept date-time;]
    [stop-accept date-time;]
}];
```

where *auth-key* is a 1- to 16-character string in double quotes, *id-number* is an integer between 1 and 255, and *date-time* is in the format YYYY/MM/DD HH:MM. (If any time fields are used, all are required.)

If no value is given for the time ranges, the default values are:

- key is always generated
- key is always accepted

Thus, if you always want your key to be accepted, simply specify a sequence such as:

```
auth md5 key "mykeyone" id 1;
```

To specify multiple MD5 keys on an interface, add the following to the appropriate OSPF interface statements:

```
auth md5 {
    md5-key
    md5-key
        .
        .
        .
    md5-key
} ;
```

where *md5-key* is as specified above.

For example, two routers may start out generating key 1 and want to switch to key 2 at 6:00 GMT. In order to make the transition between keys easier, the routers agree to stop generating key 1 at 6:00 GMT but accept key 1 until 6:10 GMT. Key 2 is accepted ten minutes before the planned switch time (i.e., 5:50 GMT). These overlapping ranges allow the clocks on the routers to be slightly out of sync. This sequence of keys would be specified by:

```
auth md5 {
    key "mykeyone" id 1 {
        stop-generate 2001/05/01 06:00;
        stop-accept 2001/05/01 06:10;
    };
    key "mykeytwo" id 2 {
        start-generate 2001/05/01 06:00;
        start-accept 2001/05/01 05:50;
    };
};
```

Chapter 13
Intermediate System to Intermediate System (IS-IS)

13.1 Overview

IS-IS is a link state interior gateway protocol (IGP), or Intra-Domain Routing Protocol, originally developed for routing International Organization for Standardization/Connectionless Network Protocol (ISO/CLNP) packets. The version distributed with GateD routes IP packets.

In ISO terminology, a router is referred to as an "intermediate system" (IS). IS-IS intra-domain routing is organized hierarchically so that a large domain can be administratively divided into smaller areas. These areas route using Level 1 intermediate systems within areas and Level 2 intermediate systems between areas. Routing between administrative domains is handled by Border Intermediate Systems (BISs) using IDRP, the Inter-Domain Routing Protocol. Level 1 systems route directly to systems within their own area. Level 1 systems route toward a Level 2 intermediate system when the destination system is in a different area. Level 2 intermediate systems route between areas and keep track of the paths to destination areas. Systems in the Level 2 subdomain route toward a destination area or another routing domain.

As with any Internet routing protocol, IS-IS support for large routing domains can also include support for many types of individual subnetworks. These subnetworks can include point-to-point links, multipoint links, and dynamically established data links as in X.25 subnetworks and broadcast subnetworks like ISO 8802 LANs. In IS-IS, all subnetwork types are treated by the subnetwork independent functions as though they were connectionless subnetworks using subnetwork convergence functions where necessary.

Like OSPF, IS-IS uses a "shortest-path first" algorithm to determine routes. A congestion control component monitors and prevents buffer deadlock at each intermediate system. GateD configuration syntax allows as much autoconfiguration as possible, thus reducing the probability of error. GateD configuration syntax also allows a policy to be specified for exchanging routing information with other protocols running in GateD, such as BGP and RIP. The IS-IS protocol supports multipath (load-split) forwarding. IS-IS also supports full injection of exterior network prefixes and attribute information, thus eliminating the need for any internal BGP or similar protocols. IS-IS supports static routing domain information at Level 2 intermediate systems.

The reachable address prefix indicates that any Network Service Access Points (NSAPs) that match the prefix can be reachable via the Subnet Point of Attachment (SNPA) with which the prefix is associated. Where the subnetwork to which this SNPA is connected is a general topology subnetwork supporting dynamically established data links, the prefix also has associated with it the required subnetwork addressing information, or an indication that it can be derived from the destination NSAP address. The address prefixes are handled by the

Level 2 routing algorithm in the same way that information about Level 1 is handled within the domain.

An API for the origination of Traffic Engineering (TE) information is available in this release. For more information on this API (and TE support in GateD), see the IS-IS TE API document. The extended metrics (larger than 63) are originated using the **extended-metrics** keyword, which defaults to off.

The ISIS implementation in GateD requires either access to the physical layer (e.g. Ethernet) or an ISO stack in order to send and receive packets. PDU (Protocol Data Units) in ISIS are sent directly over the physical layer using 802.2 LLC encapsulation. The supported operating systems with ISIS in this release are NetBSD, BSD/OS, and Linux 2.4. These systems allow either access to an AF_ISO socket or direct access to the physical layer through a PF_PACKET socket.

13.2 IS-IS Syntax

```
isis ( on | off ) {
    [ area D.D.D.D | HH.HHHH.HHHH.HHHH.HHHH.HHHH ; ]
    area auth simple key ;
    area auth md5 key key ;
    area auth {
        [ simple key ; ]
        [ md5 key key ; ]
        [ md5 key key {
            [ start-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ start-generate YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-generate YYYY/MM/DD HH:MM [.ss] ; ]
        } ; ]
    } ;
    domain auth simple key ;
    domain auth md5 key key ;
    domain auth {
        [ simple key ; ]
        [ md5 key key ; ]
        [ md5 key key {
            [ start-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ start-generate YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-generate YYYY/MM/DD HH:MM [.ss] ; ]
        } ; ]
```

```
} ;
[ domain-wide ( on | off ) ; ]
[ export-defaults metric-type ( internal | external ); ]
[ export-defaults metric ( metricnum | inherit ); ]
[ export-defaults level ( 1 | 2 ); ]
[ extended-metrics ( on | off ) ; ]
[ rfc1195-metrics ( on | off ) ; ]
[ external preference preferencenum  ; ]
[ hostname "name" ; ]
[ inet ( on | off ) ; ]
[ inet6 ( on | off ) ; ]
[ interface interface_name [ {
    [ ( enable | disable ) ; ]
    auth simple key [ level ( 1 | 2 | 1 and 2 ) ] ;
    auth md5 key key ;
    auth {
        [ simple key ; ]
        [ md5 key key ; ]
        [ md5 key key  {
            [ start-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-accept YYYY/MM/DD HH:MM [.ss] ; ]
            [ start-generate YYYY/MM/DD HH:MM [.ss] ; ]
            [ stop-generate YYYY/MM/DD HH:MM [.ss] ; ]
        }; ]
        [ level ( 1 | 2 | 1 and 2 ) ; ]
    }
    [ csn-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ] ; ]
    [ dis-hello-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ]
       ; ]
    [ hello-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ] ; ]
    [ hello-multiplier multipliernum [ level ( 1 | 2 | 1 and 2 ) ]
       ; ]
    [ lsp-interval msintervalnum ; ]
    [ level ( 1 | 2 | 1 and 2 ) ; ]
    [ max-burst burstnum ; ]
    [ mesh-blocked ; ]
    [ mesh-group meshnum ; ]
    [ metric metricnum [ level ( 1 | 2 | 1 and 2 ) ] ; ]
```

```
            [ passive ( on | off ) ; ]
            [ periodic-csn ( on | off ) ; ]
            [ priority prioritynum [ level ( 1 | 2 | 1 and 2 ) ] ; ]
            [ retransmit-interval intervalnum ; ]
        } ] ; ]
        [ level ( 1 | 2 | 1 and 2 ) ; ]
        [ overload-bit ( on | off ) ; ]
        [ preference preferencenum ; ]
        [ psn-interval intervalnum ; ]
        [ require-snp-auth ( on | off ) ; ]
        [ ribs ( unicast | unicast multicast ) ; ]
        [ spf-interval intervalnum ; ]
        [ summary-originate [ inet ] {
            [ ipv4-network mask  ipv4-netmask metric cost-value ; ]
            [ ipv4-network masklen ipv4-masklen metric cost-value ; ]
        } ; ]
        [ summary-filter [ inet ] {
            [ ipv4-network mask ipv4-netmask ; ]
            [ ipv4-network masklen ipv4-masklen ; ]
        } ; ]
        [ summary-originate inet6 {
            [ ipv6-network mask  ipv6-netmask metric cost-value ; ]
            [ ipv6-network masklen ipv6-masklen metric cost-value ; ]
        } ; ]
        [ summary-filter inet6  {
            [ ipv6-network mask ipv6-netmask ; ]
            [ ipv6-network masklen ipv6-masklen ; ]
        } ; ]
        [ systemid (D.D.D.D | HHHH.HHHH.HHHH) ; ]
        [ traceoptions isis_traceoptions ; ]
```

OSI-specific:

```
        [ config-time seconds ; ]
        [ es-config-time seconds ; ]
        [ hold-time seconds ; ]
    } ;
```

Notes:

- The interface level is restricted by the global level.

- IPv6 related options (**inet6** for example) will fail to parse unless IPv6 is supported in the code base and underlying operating system.
- ISO is the only supported encapsulation type.
- IS-IS extended reachability TLV's may be originated using **extended-metrics on**. This option must be used if metrics larger than 63 are to be configured.

See page 151 of the *Command Reference Guide* for specific information about each command.

13.3 IS-IS Defaults

```
isis off {
    config-time 60;
    es-config-time 60;
    export-defaults metric-type internal;
    export defaults metric inherit;
    export defaults level 2;
    extended-metrics off;
    rfc1195-metrics on;
    external preference 151;
    hold-time 120;
    inet on;
    inet6 off;
    interface all {
        enable;
        csn-interval 10 level 1 and 2;
        dis-hello-interval 3 level 1 and 2;
        hello-interval 10 level 1 and 2;
        hello-multiplier 3 level 1 and 2;
        level 1 and 2;
        lsp-interval 33;
        max-burst 5;
        metric 10 level 1 and 2;
        periodic-csn off;
        priority 64 level 1 and 2;
        retransmit-interval 5;
    };
    level 1 and 2;
    overload-bit off;
    preference 11;
```

```
        psn-interval 2;
        require-snp-auth off;
        ribs unicast;
        spf-interval 2;
        traceoptions none;
    }
```

13.4 IS-IS Sample Configurations

Example 1

Export all static routes into level 2 external reachability.

```
    export proto isis metric-type external level 2 {
        proto static {
            all ;
        } ;
    } ;
```

Example 2

Export all IS-IS external routes into OSPF ASE with metric 2 type 1.

```
    export proto ospfase type 1 metric 2 {
        proto isis external {
            all ;
        } ;
    } ;
```

Chapter 14
Border Gateway Protocol (BGP)

14.1 BGP Overview

The Border Gateway Protocol (BGP) is an exterior, or inter-domain, routing protocol used for exchanging routing information between autonomous systems. BGP is used to exchange routing information between multiple transit autonomous systems as well as between transit and stub autonomous systems. BGP uses path attributes to provide more information about each route and in particular to maintain an autonomous system (AS) path. An AS path includes the AS number of each autonomous system that the route has transited, which provides information sufficient to prevent routing loops in an arbitrary topology. Path attributes may also be used to distinguish between groups of routes to determine administrative preferences, allowing greater flexibility in determining route preference to achieve a variety of administrative ends. GateD supports version 4 of the BGP protocol.

BGP supports two basic types of sessions between neighbors: internal (sometimes referred to as IBGP) and external (EBGP). Internal sessions are run between routers in the same autonomous system. External sessions run between routers in different autonomous systems. When an AS sends routes to an external peer, the local AS number is prepended to the AS path. This means that routes received from an external peer are guaranteed to have the AS number of that peer at the start of the path. In general, routes received from an internal neighbor will not have the local AS number prepended to the AS path. Those routes will have the same AS path that the route had when the originating internal neighbor received the route from an external peer. Routes with no AS numbers in the path may be legitimately received from internal neighbors. These routes should be considered internal to the receiver's own AS.

External BGP sessions may or may not include a single metric, which BGP calls the Multi-Exit Discriminator (MED) among the path attributes. MED is a 32-bit unsigned integer. Smaller values of the MED are preferred. This metric is used only to break ties between routes with equal preference from the same neighboring AS.

Internal BGP sessions carry at least one metric in the path attributes, which BGP calls the **`Local_Pref`**. A route is preferred if its value for this metric is larger. Internal sessions may optionally include a second metric, the MED, carried in from external sessions. The use of these metrics is dependent on the type of internal protocol processing that is specified.

BGP collapses as many routes with similar path attributes as it can into a single update for advertisement. It also sends another update once the maximum packet size is reached. The churn caused by the loss of a neighbor will be minimized, and the initial advertisement sent during peer establishment will be maximally compressed. BGP does not read information from the kernel message by message, but fills the input buffer. BGP processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all

incoming data queued on the socket. This feature may cause other protocols to be blocked for prolonged intervals by a busy peer connection. All unreachable messages are collected into a single message and sent prior to reachable routes during a flash update. Another update is sent once the maximum packet size is reached.

Two internal routing groups exist: **group type internal** and **group type routing**. The **group type internal** expects all peers to be directly attached to a shared subnet so that, like external peers, the next hops received in BGP advertisements may be used directly for forwarding. But **group type routing** will determine the immediate next hops for routes by using the next hop received with a route from a peer, and using this next hop to look up an immediate next hop in an IGP's routes. Such groups support distant peers but need to be informed of the IGP whose routes they are using to determine immediate next hops.

For **group type internal** BGP, where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group. The copy includes possible adjustments to the next-hop field as appropriate to each peer. Another update is sent once the maximum packet size is reached. This process minimizes the computational load of running large numbers of peers in these types of groups.

BGP allows unconfigured peers to connect if an appropriate group has been configured with an **allow** clause.

14.2 BGP Syntax

```
bgp ( on | off )
    {
    [ clusterid host-id ; ]
    [ defaultmetric metric ; ]
    [ discard-nonprefixed-confederations |
      ignore-nonprefixed-confederations ]
    [ open-on-accept ; ]
    [ preference bgppreference ; ]
    [ traceoptions trace_options ; ]
    group type
        ( external peeras autonomoussystem
        | internal peeras autonomoussystem
        | routing peeras autonomoussystem proto protocol )
            [ ascount count ]              # external only
            [ comm community_values ]
            [ confed ]
            [ gateway host ]
            [ holdtime time ]
            [ ignorefirstashop ]           # external only
            [ keep ( all | none ) ]
```

```
        [ keepalivesalways ]
        [ localtcp local_address ]
        [ localas autonomous_system ]   # external only
        [ logupdown ]
        [ med ]
        [ metricout metric ]
        [ nexthopself ]                 # external only
        [ no-mp-nexthop ]
        [ noaggregatorid ]
        [ nogendefault ]
        [ nov4asloop ]
        [ outdelay time ]               # external only
        [ passive ]
        [ preference grouppreference ]
        [ preference2 grouppreference2 ]
        [ recvbuffer buffer_size ]
        [ reflector-client [ no-client-reflect ] ]
        # internal and routing types only
        [ routetopeer ]
        [ sendbuffer buffer_size ]
        [ setpref metric ]      # internal and routing types only
        [ showwarnings ]
        [ traceoptions trace_options ]
        [ ttl ttl ]                     # routing only
{
        #
        # There can be zero or one
        # "allow" clauses within a peer group.
        #
        allow {
            all ;
            | host ipnumber ;
            | classful network ;
            | network mask mask ;
            | network masklen number ;
            | network '/' number ;
        } ;
        #
```

```
                # There can be zero or more
                # "peer" clauses within a peer group.
                #
                peer host
                    [ ascount count ]
                    [ gateway host ]
                    [ holdtime time ]
                    [ ignorefirstashop ]
                    [ keep ( all | none ) ]
                    [ keepalivesalways ]
                    [ localtcp local_address ]
                    [ logupdown ]
                    [ med ]
                    [ metricout metric ]
                    [ nexthopself ]
                    [ no-mp-nexthop ]
                    [ noaggregatorid ]
                    [ nogendefault ]
                    [ nov4asloop ]
                    [ outdelay time ]
                    [ passive ]
                    [ preference peerpreference ]
                    [ preference2 peerpreference2 ]
                    [ recvbuffer buffer_size ]
                    [ reflector-client [ no-client-reflect ] ]
                    [ routetopeer ]
                    [ sendbuffer buffer_size ]
                    [ showwarnings ]
                    [ traceoptions trace_options ]
                    [ ttl ttl ]
                ;
                #
                # There should be at least one "allow" or "peer" clause
                # within a "group type" statement.
                #
        } ;
    } ;
```

Notes:

1. You must specify the Autonomous System and Router ID at the top of your configuration file in order for BGP to work.
2. One or more **group type** clauses must appear within the bgp statement. The **group type** clause is used to create peer groups that share common attributes.
3. Within the group type statement, the **allow** and/or **peer** clauses are used to specify the peers that are in the group.
4. The **allow** clause allows peering sessions to be established by hosts within one or more networks. The **allow** clause should appear at most only once within a peering group.
5. The **peer** clause is used to individually specify peers and to permit overriding specific peering group options.
6. The **peer** clause can appear multiple times within a peering group.

More detailed descriptions of these commands can be found on page 225 of the *Command Reference Guide*. See "Examples of Importation into Multicast RIBs" on page 142 for more information about importing and BGP. See "Exporting to BGP" on page 149 for more information about exporting and BGP.

14.3 Extended BGP-4 Features

The following features are provided in extended BGP-4:

- AS Path prepend - GateD allows the prepending of Autonomous Systems.
- Route reflection (RFC 2796) - Route Reflection is supported for reduction of large internal peer groups. See "Route Reflection Overview and Examples" on page 71.
- BGP Route Flap Damping (RFC 2439) - GateD supports the varied parameters on Route Flap Damping.
- Community Support (RFC 1997) - GateD allows for filtering of routes based on communities on import. In exporting routes, GateD allows the communities to be added or deleted. See "Communities Overview and Examples" on page 77.
- BGP Confederations (RFC 3065) - BGP Confederations allows multiple internal ASs to be used for scaling large networks. This confederation is represented as a single external AS.

14.4 Route Selection

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors.

14.5 Cisco® Interoperability

GateD configuration differs greatly from Cisco® routers. This section compares the following:

- BGP route selection
- Local_Pref configuration
- MED configuration
- import and export policy configuration

14.5.1 Cisco® vs. GateD Route Selection

The following table compares Cisco® 11.0/12.0 and GateD bgp-4-16 draft route selection policy:

Cisco® (11.0/12.0)	GateD (bgp-4-16 draft)
Active Route - If the next hop is inaccessible, do not consider it.	Active Route - If GateD cannot install a route in the kernel, GateD will not consider it (select the route as the active route).
Configured Policy - Consider larger BGP administrative weights first.	Configured Policy - Consider the route with smallest preference, as determined by the policy defined in `gated.conf`. Ties are broken by the **`preference2`** with the lowest values.
Local_Pref - If the routes have the same weight, consider the route with higher local preference.	Local_Pref - If the BGP Preferences match (**`preference`** and **`preference2`**), prefer the route with the highest BGP local preference.
Local Router - If the routes have the same local preference, prefer the route that the local router originated.	
Shortest AS Path - If no route is originated, prefer the shorter AS path.	Shortest AS Path - If the routes have the same BGP local preference, prefer the route with the fewest Autonomous Systems listed in its AS path.
IGP < EGP < Incomplete - If all routes have paths with the same AS path length, prefer the lowest origin code (IGP < EGP < Incomplete).	Origin IGP < EGP < Incomplete - If routes have the same AS path length, prefer the lowest origin code. Next in preference is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.

Cisco® (11.0/12.0)	GateD (bgp-4-16 draft)
MED - If origin codes are the same and all the paths are from the same Autonomous System, prefer the path with the lowest Multi Exit Discriminator (MED) metric. A missing metric is treated as zero.	MED (if not ignored) - If origin codes are the same, prefer the highest (worst) Multi-Exit Discriminator. MEDs are only compared between routes that were received from the same neighbor AS. This test is applied only if the local AS has two or more connections to a given neighbor AS. A missing metric is treated as the best (lowest) MED. MED comparison must be enabled; it is disabled by default.
External/Internal - If the MEDs are the same, prefer external paths over internal paths.	Source IGP < EBGP < IBGP - If the MEDs are the same, prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.
Closest Neighbor - If IGP synchronization is disabled and only internal paths remain, prefer the path through the closest neighbor.	Shortest IGP distance - If the IGP distances are the same, prefer the route whose next hop IP address is closer (with respect to the IGP distance)
Lowest IP Address - If the neighbors are equally close, prefer the route with the lowest IP address value for the BGP router ID.	Lowest Router ID - If the sources are the same, prefer the route whose next hop IP address is numerically lowest.

14.6 Local_Pref Configuration Example

The following configurations set a **`Local_Pref`** of 120 for peers in AS 200. Note that GateD's configuration uses the **`setpref`** command. The **`Local_Pref`** value comes from this equation: **`Local_Pref = 254 - (global protocol preference for this route) + metric`**. The global protocol preference for BGP is 170. (See "Assigning Preferences" on page 11.) In this example, we use the syntax **`setpref 36`** to specify a **`Local_Pref`** value of **`120`** (**`254-170+36 = 120`**) for BGP routes.

Cisco®:

```
router bgp 100
  network 192.168.0.0
  neighbor 192.168.1.1 remote-as 200
  neighbor 192.168.1.1 route-map set-local-pref in
route-map set-local-pref permit 10
  set local preference 120
```

GateD:

```
group type internal peeras 200 setpref 36 {  # (254-170+36) = 120
    peer 192.168.1.1;
};
```

14.6.1 MED Configuration Example

The following configurations set a metric of 127 on routes exported to AS 200.

Cisco®:

```
ip as-path access-list 1 permit .*
route-map med permit 10
match as-path 1
set metric 127
```

GateD:

```
export proto bgp as 200 {
    proto bgp aspath .* origin any {
        all metric 127;
    };
};
```

14.6.2 Import Filter Example

Cisco®:

```
router bgp 200
  neighbor 192.168.10.32 remote-as 100
  neighbor 192.168.10.32 filter-list 2 in
ip as-path access-list 2 deny _690$
ip as-path access-list 2 permit .*
```

GateD:

```
autonomoussystem 200;
routerid 192.168.10.55;
bgp on {
    group type external peeras 100 {
        peer 192.168.10.32;
    };
};
import proto bgp aspath (.* 690) origin any {
    all restrict;
};
import proto bgp aspath (.*) origin any {
    all;
};
```

14.6.3 Export Filter Example

Cisco®:

```
router bgp 200
 neighbor 192.168.10.32 remote-as 100
 neighbor 192.168.10.32 filter-list 3 out
ip as-path access-list 3 deny _400$
ip as-path access-list 3 permit .*
```

GateD:

```
autonomoussystem 200;
routerid 192.168.10.55;
bgp on {
    group type external peeras 100 {
        peer 192.168.10.32;
    };
};
export proto bgp as 100 {
    proto bgp aspath (.* 400) origin any {
        all restrict;
    };
    proto bgp aspath (.*) origin any {
        all;
    };
```

```
};
```

14.7 BGP Issues

14.7.1 Third-Party Route Advertisement

Third-party route advertisements are a special form of advertisements to peers. In particular, any advertisement that has a next hop attribute that contains an IP address different from the IP address of the peer sending the advertisement is called "third party." A third-party advertisement is legal, essentially, when the next hop that is advertised is on the network that is used for peering.

GateD, by default, performs third party route advertisements when the next hop that it is using, if used, would result in a legal third-party route advertisement. Also, by default, GateD rejects third-party route advertisements that are illegal. There are two options that can be used to alter this default behavior: **nexthopself** and **gateway**. The former deals with routes originated by GateD, the latter with both sending and receiving third-party advertisements.

nexthopself causes GateD to include a next hop of its own IP address in all advertisements to an external peer. Hence, no advertisements that GateD sends could be considered third party.

The second option, **gateway**, is meant for use in situations where the peers are not directly connected to one another. With the **gateway** option, you specify the first hop along the path to the peer. GateD will then perform third-party route advertisements as though the network shared with the gateway were really the network shared with the peer. GateD will also substitute, on received advertisements, the address of the gateway for the address of the next hop received.

The following is a sample BGP statement in which GateD turns off third-party route advertisements with respect to peer 192.168.10.1, but not with respect to 192.168.10.2.

```
bgp yes {
    group type external peeras 1 {
        peer 192.168.10.1 nexthopself;
        peer 192.168.10.2;
    };
};
```

In the preceding example, if GateD learned reachability for network 192.168.20 with a next hop of 192.168.10.100, the advertisements to peer 192.168.10.1 and peer 192.168.10.2 would differ: the advertisement to peer 192.168.10.1 would contain a next hop of the GateD box, and the advertisement to peer 192.168.10.2 would contain a next hop of 192.168.10.100.

And here is an example where the GateD box is attached to the network 192.168.10/24, but the peer is not. Note that the gateway router (192.168.10.1) must be able to forward packets to the peer (192.168.77.12).

```
bgp yes {
    group type external peeras 1 {
        peer 192.168.77.12 gateway 192.168.10.1;
```

```
        };
    };
```

In this example, GateD will ensure that all of the next hops that it advertises to its peer (192.168.77.12) are on the network shared with the gateway (192.168.10/24). Any next hops that it receives from the peer (192.168.77.12) will be replaced with the address of the gateway (192.168.10.1).

14.7.2 Determining Next Hops

In GateD, at present, there are three different cases for next hop determination: **`group type internal`**, **`group type external`**, and anything else. Modification of the next hop for **`group type external`** is covered in "Third-Party Route Advertisement" on page 70. As far as IBGP peers are concerned, the BGP specification is clear: the next hop that is sent shall be the next hop that was received.

`group type internal` is intended for peers on directly attached networks. If the peers are not on directly shared networks, **`group type routing`** should be used.

For next hop determination, **`group type routing`** uses essentially the same algorithm that external peers with the **`gateway`** option use. GateD determines which network is being used to reach the immediate next hop to its peer. It then ensures that the next hop advertised is on the same network as the immediate next hop.

14.7.3 AS Path Stuffing and Spoofing

AS Path "stuffing" or "prepending" is accomplished with the **`ascount`** command. **`ascount`** is used to bias upstream peers' route selection. (Most routers prefer routes with shorter AS Paths.)

In previous versions of BGP, the specification had not allowed the existence of looped AS Paths. Loops must be ignored in order to allow AS prepending. The **`localas`** command can be used to spoof the AS that BGP represents to a group of peers. The default AS is that configured in the **`autonomoussystem`** statement. **`localas`** provides a way to speak BGP from more than one AS.

14.7.4 Route Reflection Overview and Examples

Generally, all border routers in a single AS need to be internal peers of each other, and, in fact, all non-border routers frequently need to be internal peers of all border routers. Although this configuration is usually acceptable in small networks, it may lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports route reflection for internal peer groups. When using route reflection, the rule that a router may not readvertise routes from internal peers to other internal peers is relaxed for some routers, called "route reflectors". A typical use of route reflection might involve a "core" backbone of fully meshed routers (all the routers in the group peered directly with all other routers in the group). Some of these routers act as route reflectors for routers that are not part of the core group.

Two types of route reflection are supported: routes can be sent to all internal peers or only to internal peers that are not members of the client's group. By default, all routes received by the route reflector from a client are sent to all internal peers (including the client's

group, but not the client itself). If the **no-client-reflect** option is enabled, routes received from a route reflection client are sent only to internal peers that are not members of the client's group. In this case, the client's group must itself be fully meshed. In either case, all routes received from a non-client internal peer are sent to all route reflection clients.

Typically, a single router will act as the reflector for a set (or cluster) of clients. However, for redundancy, two or more can also be configured to be reflectors for the same cluster. In this case, a cluster ID should be selected using the **clusterid** keyword to identify all reflectors serving the cluster. Gratuitous use of multiple redundant reflectors is not advised, because it can lead to an increase in the memory required to store routes on the redundant reflectors' peers.

No special configuration is required on the route reflection clients. From a client's perspective, a route reflector is simply a normal IBGP peer. Any BGP version 4 speaker should be able to be a reflector client.

Refer to the route reflection specification document (RFC 1966) for further details. RFC 1966 can be found at:

http://www.ietf.org/rfc/rfc1966.txt

All routes received from any group member will be sent to all other internal neighbors, and all routes received from any other internal neighbors will be sent to the reflector clients. Because the route reflector forwards routes in this way, the reflector-client group need not be fully meshed. If the **no-client-reflect** option is specified, routes received from reflector clients will only be sent to internal neighbors that are not in the same group as the sending reflector client. In this case, the reflector-client group should be fully meshed. In all cases, routes received from normal internal peers will be sent to all reflector clients.

Note: It is necessary to export routes from the local AS back into the local AS when acting as a route reflector. For example, suppose that the local AS number is 2. An export statement like the following would suffice to make reflection work correctly.

```
export proto bgp as 2 {
    proto bgp as 2 {all;}; # for reflection
    # other exports
};
```

If the cluster ID is changed and GateD is reconfigured with a **SIGHUP**, all BGP sessions with reflector clients will be dropped and restarted.

Another example follows.

```
traceoptions "/var/tmp/gated.log" replace size 1000k files 3 all;
autonomous-system 64512;
routerid 192.168.11.1;
bgp yes {
    group type internal peeras 64512 reflector-client {
        peer 192.168.10.2;
        peer 192.168.10.3;
```

```
        peer 192.168.10.4;
        peer 192.168.10.5;
        peer 192.168.10.6;
    };
    group type internal peeras 64512 {
        peer 192.168.11.2;
        peer 192.168.11.3;
    };
};

static {
    default gw 172.16.0.1 retain;
};

import proto bgp as 64512 {
    all;
};

export proto bgp as 64512 {
    proto bgp as 64512 {
        all;
    };
};
```

14.7.5 Weighted Route Damping Overview, Syntax, and Defaults

The basic idea of weighted route damping is to treat routes that are being announced and withdrawn (flapping) at a rapid rate as unreachable.

If a route flaps at a low rate, it should not be suppressed at all, or suppressed only for a brief period of time. With weighted route damping, the suppression of a route or routes occurs in a manner that adapts to the frequency and duration that a particular route appears to be flapping. The more a route flaps during a period of time, the longer it will be suppressed. The adaptive characteristics of weighted route damping are controlled by a few configurable parameters.

Currently, only routes learned via BGP are subject to weighted route damping, although no protocols will announce suppressed routes. The weighted route damping configuration statement is not within the BGP statement but is a separate and distinct configuration; conceptually, it is much like **`interface`** or **`kernel`** statements. (Refer to “Chapter 7 Interface Statement” on page 23 and “Chapter 18 Kernel Interface” on page 95 for more information.)

The syntax for weighted route damping in GateD is:

```
dampen-flap {
```

```
            [ suppress-above metric ; ]
            [ reuse-below metric ; ]
            [ max-flap metric ; ]
            [ unreach-decay time ; ]
            [ reach-decay time ; ]
            [ keep-history time ; ]
        };
```

suppress-above *metric*

suppress-above is the value of the instability metric at which route suppression will take place (a route will not be installed in the FIB or announced even if it is reachable during the period that it is suppressed).

reuse-below *metric*

reuse-below is the value of the instability metric at which a suppressed route will become unsuppressed, if it is reachable but currently suppressed. The value assigned to **reuse-below** must be less than **suppress-above**.

max-flap *metric*

max-flap is the upper limit of the instability metric. This value must be greater than the larger of 1 and **suppress_above**.

Assigned to the above three parameters is a floating point number in units of flaps. Each time a route becomes unreachable, 1 is added to the current instability metric.

reach-decay *time*

reach-decay specifies the time desired for the instability metric value to reach one half of its current value when the route is reachable. This half-life value determines the rate at which the metric value is decayed. A smaller half-life value will make a suppressed route reusable sooner than a larger value.

unreach-decay *time*

unreach-decay acts the same as **reach-decay** except that it specifies the rate at which the instability metric is decayed when a route is unreachable. It should have a value greater than or equal to **reach-decay**.

keep-history *time*

keep-history specifies the period over which the route flapping history is to be maintained for a given route. The size of the configuration arrays described below is directly affected by this value.

If only **dampen-flap {};** is specified in the configuration file, then the following default values are used:

```
suppress-above = 3.0;
reuse-below = 2.0;
max-flap = 16.0;
unreach-decay = 900;
reach-decay = 300;
```

```
keep-history = 1800
```

14.7.6 Setpref/Local_Pref Overview

Note: The term "preference" as used in **setpref/Local_Pref** is not the same as each protocol's **preference** in GateD. Each protocol has a parameter, **preference**, that specifies how active routes will be selected. When a route has been learned from more than one protocol, the active route will be selected from the protocol with the lowest **preference**. Each protocol has a default preference in this selection. **setpref/Local_Pref** is BGP-specific and does not influence how active routes from BGP will compare to those learned from other protocols.

The **setpref** option allows GateD to set the **Local_Pref** to reflect GateD's own internal preference for the route, as given by the global protocol preference value (which can be found at "Preference Selection Precedence" on page 12). **Local_Pref** can be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. The **setpref** option can be used with routing or internal type groups. The **Local_Pref** is never set directly, but rather as a function of the GateD **preference** and **setpref** metrics.

If the **setpref** option is set on one internal peer group, it must be set on all internal peer groups. The **setpref** option may be used only on internal peer group types (internal or routing).

The translation of GateD's internal preference to and from **Local_Pref** is done as follows. In the table below, *metric* is the argument to **setpref**. (For example, in the statement, "**setpref** *100*," *metric* is 100.) "Exported Preference" is the GateD preference of the exported route. "Imported Preference" is the GateD preference assigned to the imported route.

Exported Preference	Local_Pref	Imported Preference
Less than *metric*	254	*metric*
metric to 254	254 to *metric*	*metric* to 254
N/A	Greater than 254	*metric*

In effect, any GateD preference of less than *metric* is exported such that it will be re-imported (by a distant GateD) with a preference of exactly *metric*. Any preference of *metric* or above will be exported such that it will be re-imported with the same preference it had originally.

Local_Pref, as exported to BGP peers, is calculated as:

Local_Pref = 254 - (global protocol **preference** for this route) + *metric*

A value greater than 254 will be reset to 254. GateD will only send **Local_Pref** values between 0 and 254. For example, suppose GateD is sending routes to an internal group using "**setpref** *100*," and the routes are subsequently received by another router in the group, also using "**setpref** *100*."

The table below lists some sample route preferences, the **Local_Prefs** with which the routes will be sent, and the preferences with which the routes will be imported.

Preference Before Export	Local_Pref	Preference After Import
170	184=(254-170+100)	170
171	183	171
254	100	254
100	254	100
5	254	100

Notes:

- Non-GateD IBGP implementations may send **Local_Prefs** that are greater than 254. When operating a mixed network of this type, it is recommended that all routers restrict themselves to sending **Local_Prefs** in the range *metric* to 254.
- All routers in the same network that are running GateD and participating in IBGP should use **setpref** uniformly. That is, if one router has **setpref** set, all should set it, and all should use the same value of *metric*. The value for *metric* should be selected to be consistent with the import policy in use in the network. For example, if import policy sets GateD preferences ranging from 170 to 200, a **setpref** *metric* of 170 would make sense. It is advisable to set *metric* high enough to avoid conflicts between BGP routes and IGP or static routes.

Routes propagated by IBGP must include a **Local_Pref** attribute. **Local_Pref** may be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. Unless the **setpref** option has been set, BGP sends the **Local_Pref** path attribute as 100.

GateD always uses the received **Local_Pref** to select between BGP routes that have the same GateD preference. BGP routes with a larger **Local_Pref** are preferred.

For this topology:

The following configuration will cause AS 65100 to prefer routes from the BGP1--BGP2 link.

BGP1 Configuration

```
bgp yes {
    group type external peeras 65000 {
        peer 10.0.0.2; # BGP2
    };
    group type internal peeras 65100 setpref 100 {
        peer 192.168.10.2; # BGP3
    };
};
```

BGP3 Configuration

```
bgp yes {
    group type external peeras 65000 {
        peer 10.0.0.2; # BGP2
    };
    group type internal peeras 65100 setpref 99 {
        peer 192.168.10.1; # BGP1
    };
};
```

14.7.7 Communities Overview and Examples

The community attribute allows the administrator of a routing domain to tag groups of routes with a community tag. Using communities allows the administrator to limit the routes that can be imported or exported. The tag consists of 2 octets of AS and 2 octets of community ID. The **community** attribute is passed from routing domain to routing domain to maintain the grouping of these routes. A set of routes may have more than one community tag in its **community** attribute.

The import and export policy of a community is configured using the **comm** clause (or **comm-add** clause) on the **group**, **import**, and **export** statements.

Please refer to the communities specification (RFC 1997) and its accompanying usage document (RFC 1998) for further details on BGP communities. RFC 1997 can be found at:

http://www.ietf.org/rfc/rfc1997.txt

RFC 1998 can be found at:

http://www.ietf.org/rfc/rfc1998.txt

Communities can be specified as an AS and a community ID (with the **comm-split** keyword) or as one of the distinguished special communities (with the **comm** keyword). When originating BGP communities, the set of communities that is actually sent is the union of the communities received with the route (if any), those specified in group policy (if any), and those specified in export policy (if any). When receiving BGP communities, the update is matched only if all communities specified in **comm** are present in the BGP update. (If additional communities are also present in the update, it will still be matched.) The limit of 25 communities in any single policy clause can be increased at compile time by increasing the value of **AS_COMM_MAX**.

comm-split *autonomous_system community_id*

comm-split causes a community "tag" to be added to the transmitted path attributes. The *autonomous_system* part of the community should be set to the local AS, unless there is a specific need to do otherwise. This associates an AS with a community.

community no-export

community no-export is a special community that indicates that the routes associated with this attribute must not be advertised outside a BGP AS boundary.

community no-advertise

community no-advertise is a special community that indicates that the routes associated with this attribute must not be advertised to other BGP peers.

community no-export-subconfed

community no-export-subconfed is a special community that indicates that the routes associated with this attribute must not be advertised to external BGP peers.

community none

community none is not actually a community, but rather a keyword that specifies that a received BGP update is only to be matched if no communities are present. It has no effect when originating communities.

The following example will import only routes from AS 203 that are stamped with community 99:

```
import proto bgp as 203
    comm {
        comm-split 203 99
    }
{
```

```
        all;
    };
```

The following example will export only routes to AS 205 and from AS 203 that are stamped with community 99:

```
    export proto bgp as 205
        comm {
            comm-split 203 99
        }
    {
        proto bgp static {
            all;
        };
    };
```

Communities are added to a route on export with the **comm-add** aspath options.

```
    export proto bgp as 205
        comm-add {
            comm-split 203 99
        }
    {
        proto bgp static {
            all;
        };
    };
```

14.7.8 Multi-Exit Discriminator Overview and Examples

The Multi-Exit Discriminator (MED) allows the administrator of a routing domain to choose between various exits from a neighboring AS. This attribute is used only for decision-making in choosing the best route to the neighboring AS. If all the other factors for a path to a given AS are equal, the path with the lower MED value takes preference over other paths.

This attribute, if learned from an external AS, can be propagated only to internal peers, unless you are in a BGP Confederation. The MED value can be propagated to BGP Confederation external peers. The MED value is propagated only if the **med** keyword is specified on the BGP peers or group.

The MED attribute for BGP version 4 is a four-byte unsigned integer. MED is originated using the **metricout** option of group or peer statements or the **metric** option of the export statement. It is imported using the **med** keyword on the BGP group statement.

The **metricout** and **metric** options are used to specify the value of MED for exported routes. The **metricout** option can be specified on the group statement:

```
group type external peeras 31337 metricout 5 {
    peer 192.168.10.32;
    peer 192.168.10.33;
};
```

It can also be specified on the peer statement:

```
group type external peeras 31337 {
    peer 192.168.10.32 metricout 2;
    peer 192.168.10.33 metricout 3;
};
```

The equivalent metric keyword can be specified on the export statement like this:

```
export proto bgp as 31337 metric 5 {
    proto static {
        all;
    };
};
```

And like this:

```
export proto bgp as 31337 {
    proto bgp as 64000 metric 1 {
        all;
    };
    proto static metric 3 {
        all;
    };
    proto direct metric 7 {
        all;
    };
};
```

The **med** keyword must be specified on the group statement for GateD to consider metrics when calculating a next hop (the default action is to ignore MEDs).

14.7.9 Confederations

The BGP specification requires that all internal BGP speakers maintain a full mesh. As the number of BGP speakers in an AS grows, the number of peering sessions that must be maintained grows factorially. This can put a great strain on infrastructure both in terms of the hardware in routers and in terms of the amount of bandwidth consumed by routing traffic.

In order to help relieve the strain on resources, RFC 3065 specifies an alternative to full mesh IBGP known as "BGP Confederations." A BGP Confederation is a collection of autono-

mous systems that present themselves as a single AS to peers outside of the confederation. RFC 3065 can be found at:

http://www.ietf.org/rfc/rfc3065.txt

All BGP speakers within a confederation are assigned two AS numbers. The first of these is their normal AS number to be used within the confederation. The second is known as their confederation ID.

All BGP speakers within a single confederation must be assigned the same confederation ID. This confederation ID is the autonomous system number that BGP speakers outside of the confederation see as consisting of all BGP speakers within the confederation, despite the fact that the various members of the confederation can be within different autonomous systems.

When BGP speakers within the same confederation communicate with each other, they perform identically to BGP speakers not in confederations with one exception: rather than using the **AS_SEQUENCE** and **AS_SET** path attributes, they use the **CONFED_SEQUENCE** and **CONFED_SET** path attributes. Then, when a BGP speaker within the confederation goes to advertise routes to a BGP speaker not within the confederation, all attributes of the type **CONFED_SEQUENCE** or **CONFED_SET** are stripped and replaced with a single **AS_SEQUENCE** consisting of the confederation identifier. In this fashion, the internal AS topology of the confederation is kept hidden from the rest of the world.

The following **gated.conf** shows a confederation border router. It has two peers outside of the confederation, one inside the confederation, and some confederation internal peers.

```
autonomoussystem 64512;
confed-id 100;
bgp yes {
    group type routing peeras 64512 confed proto ospf {
        peer 192.168.1.1 ;
        peer 192.168.1.4 ;
    } ;
    group type external peeras 65000 confed {
        peer 10.132.10.1 ;
    } ;
    group type external peeras 200 {
        peer 172.16.50.1 ;
    } ;
} ;

# Import everything from our internal confederation peers
    import proto bgp as 64512 {
        all ;
    } ;
```

```
# Import everything from our external confederation peer
    import proto bgp as 65000 {
        all ;
    } ;

# Import everything from our external non-confederation peer
    import proto bgp as 200 {
        all ;
    } ;

# Redistribute everything from our external non-confederation and our
# external confederation peer to our internal peers. Note that we are
# not operating as a route reflector, so we do not redistribute routes
# from our internal peers to our other internal peers.

    export proto bgp as 64512 {
        proto bgp as 200 {
            all ;
        } ;
        proto bgp as 65000 {
            all ;
        } ;
    } ;

# Redistribute our routes from our external confederation peer to our
# internal confederation peers and our external non-confederation
# peer.
    export proto bgp as 65000 {
        proto bgp as 200 {
            all ;
        } ;
        proto bgp as 64512 {
            all ;
        } ;
    } ;

# We want to receive traffic for this AS on our external links, so
# propogate everything from our confederation.

    export bgp as 200 {
```

```
        proto bgp as 64512 {
            all ;
        } ;
        proto bgp as 65000 {
            all ;
        } ;
    } ;
```

Chapter 15
Router Discovery

15.1 Router Discovery Overview

The Router Discovery Protocol is an IETF standard protocol, RFC 1256, used to inform hosts of the existence of routers. It is intended to be used instead of having hosts wiretap routing protocols, such as RIP. It is used in place of, or in addition to, statically-configured default routes in hosts. RFC 1256 can be found at:
http://ietf.org/rfc/rfc1256.txt

The protocol is split into two portions: the **`server`** portion, which runs on routers, and the **`client`** portion, which runs on hosts. GateD treats these much like two separate protocols, only one of which can be enabled at a time.

15.1.1 The Router Discovery Server

The router discovery server runs on routers and announces their existence to hosts. It announces the routers' existence by periodically multicasting or broadcasting a router advertisement from each interface on which it is enabled. These router advertisements contain a list of all the routers' addresses on a given interface, and the preferences indicate which address or addresses are a better choice for use as a default route.

Initially, these router advertisements occur every few seconds, then fall back to every few minutes. In addition, a host can send a router solicitation to which the router will respond with a unicast router advertisement (unless a multicast or broadcast advertisement is due momentarily).

Each router advertisement contains an advertisement **`lifetime`** field indicating for how long the advertised addresses are valid. This lifetime is configured such that another router advertisement will be sent before the lifetime has expired. A lifetime of zero is used to indicate that one or more addresses are no longer valid.

On systems supporting IP multicasting, the router advertisements are, by default, sent to the all-hosts multicast address **`224.0.0.1`**. However, the use of **`broadcast`** can be specified. When router advertisements are being sent to the all-hosts multicast address, or an interface is configured for the limited-broadcast address **`255.255.255.255`**, all IP addresses configured on the physical interface are included in the router advertisement. When the router advertisements are being sent to a net or subnet broadcast, only the address associated with that net or subnet is included.

A host listens for router advertisements via the all-hosts multicast address (**`224.0.0.1`**) if IP multicasting is available and enabled, or on the interface's broadcast address. When starting

up, or when reconfigured, a host can send a few router solicitations to the all-routers multicast address, **224.0.0.2**, or the interface's broadcast address.

When a router advertisement with non-zero lifetime is received, the host installs a default route to each of the advertised addresses. If the preference is ineligible, or the address is not on an attached interface, the route is marked unusable but retained. If the preference is usable, the metric is set as a function of the preference such that the route with the best preference is used. If more than one address with the same preference is received, the one with the lowest IP address will be used. These default routes are not exportable to other protocols.

When a router advertisement with a zero lifetime is received, the host deletes all routes with next-hop addresses learned from that interface of the sending router. In addition, any routes learned from ICMP redirects pointing to these addresses will be deleted. The same will happen when a router advertisement is not received to refresh these routes before the lifetime expires.

15.1.2 The Router Discovery Client

The router discovery client is provided for historical purposes only. It is an untested feature of GateD and will likely be deprecated in the future.

15.2 Router Discovery Syntax

```
routerdiscovery server ( on | off ) [ {
traceoptions trace_options ;
interface phys_interface_list
    [ maxadvinterval max_time ]
    [ minadvinterval min_time ]
    [ lifetime life_time ]
    ;
address interface_list
    [ advertise | ignore ]
    [ broadcast | multicast ]
    [ ineligible | preference preference ]
    ;
} ] ;

routerdiscovery client ( on | off ) [ {
    traceoptions trace_options ;
    preference preference ;
    interface phys_interface_list
        [ enable | disable ]
        [ multicast | broadcast ]
        [ quiet | solicit ]
        ;
} ] ;
```

15.3 Router Discovery Defaults

```
routerdiscovery server off  [ {
    interface all
        ( maxadvinterval 00:10:00
        minadvinterval .75*max_time
        lifetime 3*max_time )
        ;
    address interface_list
    advertise
    multicast
    preference 0
    ;
} ] ;

routerdiscovery client off  [ {
    preference 0 ;
    interface all enable
        enable
        multicast
        ;
} ] ;
```

15.4 Router Discovery Examples

15.4.1 Example 1

The following example runs router discovery on the interface fxp0, sending solicitations to the multicast address.

```
routerdiscovery client on {
        interface fxp0 enable multicast solicit;
};
```

15.4.2 Example 2

The following example runs the router discovery server on interface fxp0 sending advertisements no more often than once every minute, and no less often than once every 6 minutes. All routers that it advertises out interface fxp0 will be advertised with a lifetime of 10 minutes.

```
routerdiscovery server on {
        interface fxp0 minadvinterval 1:00 maxadvinterval 6:00
```

```
                lifetime 10:00;
        }
```

Chapter 16
Internet Control Message Protocol (ICMP)

16.1 ICMP Overview

On systems without the BSD routing socket, GateD listens to Internet Control Message Protocol (ICMP) messages received by the system. GateD currently supports **redirect**. Processing of ICMP redirect messages is handled by the **redirect** statement. (See "Chapter 17 Redirect Processing" on page 91 for more information about **redirect**.)

Currently, the only reason to specify the **icmp** statement is to be able to trace the ICMP messages that GateD receives. These messages may be traced to a separate log file as allowed by any GateD **traceoptions** clause. This allows for easy separation of non-redirect ICMP messages from redirect messages in the trace file.

16.2 ICMP Syntax

```
icmp {
    traceoptions
        [ tracefile [ replace ]
        [ size tracesize [ k | m ] files tracefiles ] ] [ nostamp ]
        [ trace_global_options | trace_protocol_packets ]
        [ except  ( trace_global_options | trace_protocol_packets ) ];
} ;
```

More detailed descriptions of these commands can be found on page 307 of the *Command Reference Guide*.

16.3 ICMP Sample Configuration

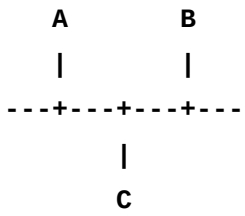
This example traces all ICMP messages received except for redirects, which may be traced from the redirect clause.

```
icmp {
    traceoptions "/tmp/icmp_log" packets routerdiscovery info error ;
}
```

Chapter 17
Redirect Processing

17.1 Redirect Overview

ICMP redirects are messages sent by a router to an originator of data, indicating that a different hop should be used to reach the destination. A router sends a redirect when a routing table lookup for a received datagram results in transmission of the datagram out the same interface on which it was received. An example is:



Node A is a host, while nodes B and C are routers. A sends a packet to some network N and uses C as a next hop. After looking up the next hop for N, if C discovers that its next hop for N is router B, C can send a redirect to A, indicating that it should use B as a next hop directly instead of C.

17.2 Why GateD Monitors Redirects

Redirects have traditionally been intended for hosts. It is expected that routers have more accurate information about the network than hosts, which are not participating in a routing protocol. Since GateD operates as a router, ICMP redirects are accepted only under certain circumstances.

Many operating systems do not allow the administrator to ignore redirects. To ignore the effects of redirects, GateD must process each one and actively monitor and change the state of the routing table.

GateD monitors ICMP redirects through a raw ICMP socket, and, where supported, a kernel routing socket. On systems without a routing socket, it may not be possible to discern the action taken by the kernel upon receipt of a redirect.

17.3 Redirect Processing

Redirects are ignored if:

- The source is not on the same network as the receiving interface.
- The source is not a router currently in use, e.g., by a route installed in the FIB.
- The source is one of our own interfaces.
- The destination being redirected matches an interface route.

- **noredirects** is configured on the receiving interface (see **redirect interface** policy).
- The source does not match **trustedgateways** policy (see **redirect gateway** policy).
- The source is not directly reachable (by an interface).

According to the IETF Router Requirements document, all ICMP redirects are processed as host redirects. If a net redirect is received, GateD attempts to update the FIB by changing the route to a host route.

If a redirect is received on the routing socket, the kernel can indicate that the redirect was installed. This allows GateD to install a "mirror" route in its own RIB. These routes are deleted after three minutes if the state is not refreshed. This allows the transient presence of a redirect route. It is expected that the IGP will provide a better route, which will override the redirect within that time period.

17.4 Configuration

Policy for receipt of redirects may be based on both the receiving interface (the **interface** option) and source gateway (the **trustedgateways** option). The preference of installed routes is given with the **preference** option.

17.5 Redirect Syntax

```
redirect on | off
[ {
    preference preference ;
    interface interface_list [ noredirects ] [ redirects ] ;
    trustedgateways gateway_list ;
    traceoptions trace_options ;
} ] ;
```

More detailed descriptions of these commands can be found on page 313 of the *Command Reference Guide*. See "Chapter 31 Route Importation" on page 137 for information about importing and redirects.

17.6 Configuration Defaults

The default configuration values are:

```
redirect on {
     preference 30;
     interface all redirects;
};
```

17.7 Redirect Sample Configurations

Example 1

This configuration disables processing of redirects on all interfaces and traces reception to the file */tmp/redirlog*.

```
redirect on {
    traceoptions "/tmp/redirlog" all;
    interface all noredirects;
};
```

Example 2

This configuration enables processing of redirects on interface '**le0**', only if they were originated by the router with interface address 192.168.10.2. Any routes created by redirect processing are given a GateD preference of 200.

```
redirect on {
    preference 200;
    interface le0 redirects;
    trustedgateways 192.168.10.2;
};
```

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.

Chapter 19
Static Routes

19.1 Static Overview

The **static** statements define the static routes used by GateD. A single **static** statement can specify any number of routes. The **static** statements occur after protocol statements and before control statements in the **gated.conf** file. Any number of **static** statements may be specified, each containing any number of static route definitions. These routes can be overridden by routes with better preference values.

19.2 Static Syntax

```
static {
    static_dest gateway gateway_list
        [ interface interface_list ]
        [ as autonomous_system ]
        [ preference preference ]
        [ static_route_flags ] ;
    static_dest interface interface
        [ preference preference ]
        [ static_route_flags ] ;
};
```

where

static_dest is:

```
host [ inet6 ] host |
[ inet6 ] default |
network [ ( mask mask ) | ( masklen number ) ]
```

and

gateway_list is one or more gateways (routers) that can be used to reach the specified host or subnet.

and

static_route_flags are:

```
( retain | reject | blackhole | noinstall | unicast | multicast )
```

and

host is a host DNS name or address

interface is:

```
( name | address | local address | remote address )
```

interface_list is:

```
all | ( interface ...)     # a list of interfaces
```

More detailed descriptions of these commands can be found on page 337 of the *GateD Command Reference Guide*.

Chapter 20
Distance Vector Multicast Routing Protocol (DVMRP)

20.1 DVMRP Overview

The **dvmrp** statement is used to configure DVMRP, which is compliant with the DVMRPv3 specification.

DVMRP is the original IP multicast routing protocol. It was designed to run over both multicast capable LANs (like Ethernet) as well as through non-multicast capable routers. In the case of non-multicast capable routers, the IP multicast packets are "tunneled" through the routers as unicast packets. Because DVMRP replicates the packets, it has an effect on performance, but has provided an intermediate solution for IP multicast routing on the Internet while router vendors decide to support native IP multicast routing.

DVMRP has both "tree construction" and "route" passage functions. The DVMRP "routes" are loaded into the multicast RIB under import policy and exported using export policy.

20.2 DVMRP Syntax

```
dvmrp ( on | off | routing-only ) {
        [ defaultmetric metric ; ]
        [ preference pref ; ]
        [ prune-lifetime time ; ]
        [ traceoptions trace_options ; ]
        [ interface interface_list {
                [ enable ; | disable ; | routing-only ; ]
                [ metric metric ; ]
                [ noretransmit ; ]
                [ tunnel-compat ; ]
                [ nodvmrpout ; ]
        }; ]
};
```

More detailed descriptions of these commands can be found on page 355 of the *Command Reference Guide*.

20.3 Sample DVMRP Configurations

20.3.1 Example 1

This configuration configures a DVMRP tunnel between 10.1.25.13 and 10.1.16.4.

```
# Simple draft-10 compliant tunnel
```

```
interfaces {
        define p2p local 10.1.25.13 remote 10.1.16.4 tunnel ipip;
};

dvmrp yes {
        interface 10.1.16.4;
};

static {
        10.1.16.0 masklen 24 gw 10.1.25.14;
};
```

20.3.2 Example 2

```
# Simple mrouted compatible tunnel
interfaces {
        define p2p local 10.1.25.13 remote 10.1.16.4 tunnel ipip;
};

dvmrp yes {
        interface 10.1.16.4 {
                tunnel-compat;
        };
};

static {
         10.1.16.0 masklen 24 gw 10.1.25.14;
};
```

20.4 DVRMP Defaults

The following configuration is equivalent to **dvmrp on**.

```
dvmrp on {
        defaultmetric 1;
        prune-lifetime 7200;
        preference 70;
        interface all {
                enable;
                metric 1;
        };
};
```

Chapter 21
Protocol Independent Multicast (PIM)

21.1 Overview

Traditional multicast routing mechanisms (for example, DVMRP and MOSPF) were intended for use within regions where groups are densely populated or bandwidth is universally plentiful. When groups, and senders to these groups, are distributed sparsely across a wide area, these "dense mode" schemes do not perform efficiently. PIM is made of two protocols, one for each type of group distribution. PIM Sparse Mode, PIM-SM, provides efficient routing for a group distributed sparsely across a wide area. PIM Dense Mode, PIM-DM, provides multicast routing for a densely populated group.

Multicasting protocols require two different functions in order to create source-based trees or group-based trees:

- a set of routes used to calculate the reverse path forwarding
- a mechanism by which to build trees

PIM is protocol independent because it depends on existing unicast routes to calculate the reverse path forwarding. In contrast, DVMRP passes this set of routes within the protocol.

There are two versions of the PIM-SM protocol. PIM-SM version 1 is documented in RFC 2117. PIM-SM version 2 was constructed to address some of the shortcomings of PIM-SM version 1. GateD implements only version 2, which is an RFC but is not considered complete enough to implement (RFC 2362). In going from draft-ietf-pim-sm-v2-new-01 to draft-ietf-pim-sm-v2-new-02, the BSR functionality was removed and placed in its own internet draft. GateD implements the PIM-SM protocol as described in draft-ietf-pim-sm-v2-new-02, but the BSR functionality as described in draft-ietf-pim-sm-v2-new-01.

GateD does not currently implement PIM-DM.

21.2 PIM Syntax

```
pim ( on | off){
  [ traceoptions trace_options ; ]
  [ hello-interval sec ; ]
  [ hello-holdtime sec ; ]
  [ hello-priority pri ; ]
  [ mrt-period sec ; ]
  [ mrt-stale-mult m ; ]
  [ assert-holdtime sec ; ]
  [ jp-interval sec ; ]
  [ jp-holdtime sec ; ]
```

```
        sparse component_name {
            [ mrt-spt-mult m ; ]
            [ threshold bps ; ]
            [ threshold-dr bps ; ]
            [ threshold-rp bps ; ]
            [ reg-sup-timeout secs ; ]
            [ probe-period secs ; ]
            [ static-rp grp-address masklen len rp-address ; ]
            [ bsr-holdtime secs ; ]
            [ wholepkt-checksum ; ]
            [ rp-switch-immediate ; ]
            [ dr-switch-immediate ; ]
            [ bsr ( off | no ) | ( address | on | yes ) [ {
                    [ priority pri ; ]
                    [ bsr-period secs ; ]
            } ] ; ]
              [ crp ( off | no ) | ( address | on | yes ) [ {
                           [ priority pri ; ]
                    [ crp-holdtime secs ; ]
                    [ crp-adv-period crp-adv-periodsecs ; ]
                    [ group {
                          [ group-address [ priority pri ] ; ]
                          [ group-address mask mask [ priority pri ] ;
                               [ group-address masklen length [ priority pri
] ; ]
                          [ all [ priority pri ] ; ]
                             [ host host [ priority pri ] ; ]
                       } ; ]
               } ] ; ]
            interface interface-list [ {
                    [ ( enable | disable ) ; ]
                    [ hello-interval sec ; ]
                    [ hello-holdtime sec ; ]
                    [ hello-priority pri ; ]
                    [ assert-holdtime sec ; ]
                    [ jp-interval sec ; ]
                    [ jp-holdtime sec ; ]
                    [ boundary ; ]
            } ] ;
         };
        };
```

More detailed information on PIM commands can be found on page 376 of the *Command Reference Guide*.

21.3 Defaults

PIM must be configured to run on at least one interface.

```
pim on {
    traceoptions none;
    mrt-spt-mult 14;
    hello-interval 30;
    hello-holdtime 105;
    hello-priority 0;
    mrt-period 15;
    mrt-stale-mult 14;
    assert-holdtime 180;
    jp-interval 60;
    jp-holdtime 210;
    sparse "sm0" {
        interface all {
            hello-interval 30;
            hello-holdtime 105;
            hello-priority 0;
            assert-holdtime 180;
            jp-interval 60;
            jp-holdtime 210;
        };
        crp no;
        bsr no;
        threshold 1000
        threshold-dr 1000
        threshold-rp 1000
        reg-sup-timeout 60
        probe-period 5
        bsr-holdtime 130
    };
};
```

21.4 PIM Tracing Options

See “Chapter 4 Trace Statements” on page 15 for generic traceoptions, and refer to below for PIM-specific tracing options.

Packet tracing options (which may be modified with **`detail`**, **`send`** or **`recv`**):

`packets -` Trace all PIM packets.
`hello -` Trace PIM router hello packets.

register - Trace PIM register and register stop packets.
bootstrap - Trace PIM bootstrap packets.
jp - Trace PIM Join/Prune packets.
assert - Trace PIM assert packets.
debug - Trace state information that is mostly of use to developers.

21.5 Examples

Example 1

The following example configures a PIM-SM component that runs on interfaces fxp0 and fxp1, and uses OSPF to determine the unicast topology.

```
ospf yes {
    defaults {
        ribs unicast multicast;
    };
    backbone {
        interface fxp0 fxp1;
        priority 1 ;
        auth simple “mypw” ;
    };
};
pim yes {
    sparse "smo" {
        interface fxp0 {
            enable;
        };
        interface fxp1 {
            enable;
        };
    };
};
```

Example 2

This is a sample use of PIM-SMv2 over RIP. On interfaces qe0, qe1, qe2, and qe3 IGMP is also running.

```
traceoptions "/var/tmp/gated.log" replace all ;
igmp yes {
    interface le0 { disable; };
    interface qe0 { enable; };
```

```
    interface qe1 { enable; };
    interface qe2 { enable; };
    interface qe3 { enable; };
};
pim yes {
    traceoptions "/var/tmp/gated.log" replace packets route;
    sparse "sm0" {
        interface le0 { disable; };
        interface qe0 { enable; };
        interface qe1 { enable; };
        interface qe2 { enable; };
        interface qe3 { enable; };
        bsr qe0 {
            priority 1;
        };
        crp qe0;
    };
};
rip yes {
    traceoptions none ;
    interface le0 noripin noripout ;
    interface qe ripout ripin version 2;
};
static {
    default gateway 198.32.4.1 preference 20 retain;    # router
   10.2.0.0 mask 255.255.255.0 gateway 10.1.0.3 preference 50 multicast
     unicast;
};
import proto rip  {
    all unicast multicast;
};
export proto rip {
    proto rip {
        all restrict;
    };
};
```

Chapter 22
Multi-Protocol - Border Gateway Protocol (MPBGP)

22.1 MPBGP Overview

The Multi-Protocol Border Gateway Protocol (MPBGP) is a set of extensions to BGP-4 to make the protocol capable of carrying routing information for IPv6 and Multicast routes. These extensions are backward compatible, making it possible for BGP-4 routers to interoperate with MPBGP routers. BGP is an exterior routing (or inter-domain routing) protocol used for exchanging routing information between autonomous systems. (See “Chapter 14 Border Gateway Protocol (BGP)” on page 61 for more information about BGP.) MPBGP will support the use of multi-protocol extensions for BGP-4, IPv6 and Multicast.

For more details on the multiple RIBs and the multicast RIB, see “Chapter 9 Multiple Routing Information Bases (RIBs)” on page 33.

MPBGP adds two new attributes, Multiprotocol Reachable NLRI (MP_REACH_NLRI) and Multi-protocol Unreachable NLRI (MP_UNREACH_NLRI). MP_REACH_NLRI carries the set of reachable destinations with next hop information. MP_UNREACH_NLRI contains the set of unreachable destinations.

Capability Advertisement is used to determine whether the Multiprotocol Extensions can be used with a particular peer. If a peer is found not to support these extensions, the MPBGP router drops the peering session.

22.2 MPBGP Syntax

Note: You must specify the AS and routerid at the top of your configuration file in order for BGP to work.

```
mpbgp ( on | off )
    {
    [ clusterid host-id ; ]
    [ defaultmetric metric ; ]
    [ discard-nonprefixed-confederations |
    ignore-nonprefixed-confederations ]
    [ open-on-accept ; ]
    [ preference mpbgppreference ; ]
    [ traceoptions trace_options ; ]
    group type
        ( external peeras autonomoussystem
        | internal peeras autonomoussystem
        | routing peeras autonomoussystem proto protocol )
```

```
        [ ascount count ]                   # external only
        [ comm community_values]
        [ confed ]
        [ gateway gateway_ip_address]
        [ holdtime time ]
        [ ignorefirstashop ]                # external only
        [ interface interface_list ]        # routing only
        [ keep ( all | none ) ]
        [ keepalivesalways ]
        [ localas autonomous_system ]       # external only
        [ localtcp local_address ]
        [ localv4addr ipv4_address ]
        [ localv6addr ipv6_address ]
        [ logupdown ]
        [ med ]
        [ metricout metric ]
        [ nexthopself ]                     # external only
        [ noaggregatorid ]
        [ nogendefault ]
        [ nov4asloop ]
        [ outdelay time ]
        [ passive ]
        [ preference grouppreference ]
        [ preference2 grouppreference2 ]
        [ recvbuffer buffer_size ]
      [ reflector-client [ no-client-reflect ] ]   # internal or
                                                   # routing only
        [ remotev4addr ipv4_address ]
        [ remotev6addr ipv6_address ]
        [ routetopeer ]
        [ sendbuffer buffer_size ]
      [ setpref metric ]                # internal or routing only
        [ showwarnings ]
        [ traceoptions trace_options ]
        [ ttl ttl ]                         # routing only
        [ caps { [v4u] | [v4m] | [v4um] | [v6u] | [v6m] | [v6um]
           | [no-caps] } ]
        #
        # There can be zero or one "allow" clauses within
        # a peer group.
        #
    {
        [inet6] allow {
                all ;
                | host ipaddress ;
                | classful_network ;
                | network mask mask ;
                   | network masklen masklennumber ;
```

```
                    | network / masklennumber ;
                } ;
            #
            # There can be zero or more "peer" clauses within
            # a peer group.
            #
            peer host
                [ ascount count ]
                [ gateway gateway ]
                [ holdtime time ]
                [ ignorefirstashop ]
                [ keep ( all | none ) ]
                [ keepalivesalways ]
                [ localtcp local_address ]
                [ localv4addr ipv4_address ]
                [ localv6addr ipv6_address ]
                [ logupdown ]
                [ med ]
                [ metricout metric ]
                [ nexthopself ]
                [ noaggregatorid ]
                [ nogendefault ]
                [ nov4asloop ]
                [ outdelay time ]
                [ passive ]
                [ preference peerpreference ]
                [ preference2 peerpreference2 ]
                [ recvbuffer buffer_size ]
                [ remotev4addr ipv4_address ]
                [ remotev6addr ipv6_address ]
                [ routetopeer ]
                [ sendbuffer buffer_size ]
                [ showwarnings ]
                [ traceoptions trace_options ]
                [ ttl ttl ]
            } ;
        #
        # There should be at least one "allow" or "peer"
        # clause within a "group type" statement.
        #
    } ;
} ;
```

More detailed descriptions of these commands can be found on page 429 of the *Command Reference Guide*.

22.3 MPBGP Configurable Options

See the following sections for more information about specific MPBGP options.

22.3.1 Route Reflection

MPBGP supports route reflection for internal peer groups. When using route reflection, the rule that a router may not readvertise routes from internal peers to other internal peers is relaxed for some routers, called "route reflectors." See "Route Reflection Overview and Examples" on page 71 for more information about route reflection.

22.3.2 Weighted Route Damping

The basic idea of weighted route damping is to treat routes that are being announced and withdrawn (flapping) at a rapid rate as unreachable. See "Weighted Route Damping Overview, Syntax, and Defaults" on page 73 for more information about weighted route damping.

22.3.3 Setpref/Local_Pref

The **`setpref`** option allows GateD to set the **`Local_Pref`** to reflect GateD's own internal preference for the route, as given by the global protocol preference value. **`Local_Pref`** may be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. See "Setpref/Local_Pref Overview" on page 75 for more information about **`setpref`**.

22.3.4 Communities and Extended Communities

The communities attribute allows the administrator of a routing domain to tag groups of routes with a community tag. Using communities allows the administrator to limit the routes that can be imported or exported. See "Communities Overview and Examples" on page 77 for more information about communities.

22.3.5 Multi-Exit Discriminator

The Multi-Exit Discriminator, or MED, allows the administrator of a routing domain to choose between various exits from a peering autonomous system (AS). See "Multi-Exit Discriminator Overview and Examples" on page 79 for more information about Multi-Exit Discriminator.

22.3.6 Confederations

A BGP Confederation is a collection of ASs that present themselves as a single AS to peers outside of the confederation. See "Confederations" on page 80 for more information about Confederations.

Chapter 23
Multicast Source Discovery Protocol (MSDP)

23.1 MSDP Overview

MSDP is intended to join administratively separate PIM-SM regions by distributing information about multicast sources within each region. MSDP speakers peer over TCP connections and announce or forward information about sources and the groups to which they are multicasting. When a rendezvous point in one PIM-SM domain learns (via MSDP) of a multicast source in another PIM-SM domain, it then attempts to join toward the multicast tree rooted at the source.

MSDP is designed to work very closely with PIM-SM. In order for a PIM-SM component to learn of sources from MSDP, and in order for MSDP to propagate information about sources for the PIM-SM domain in which it resides, a PIM-SM component must be explicitly associated with MSDP.

23.2 MSDP Syntax

```
msdp ( on | off ) {
    [ traceoptions trace_options ; ]
    [ peer local_host remote_host mesh id peeras asnum ; ]
    [ connect-retry-period sec ; ]
    [ keepalive-period sec ; ]
    [ peer-holdtime sec ; ]
    [ sa-cache-timeout sec ; ]
    [ sa-holddown sec ; ]
    [ sa-filter [ import | export ] source_ip masklen length group_ip
        masklen length ; ]
    [ pim-filter source_ip masklen length group_ip masklen length ; ]
    [ static-rpf-peer peer_ip rp_addr_ip ; ]
    [ default-rpf-peer peer_ip ; ]
    [ msdp-draft-6-compat ; ]
} ;
```

More detailed descriptions of these commands can be found on page 505 of the *Command Reference Guide*.

23.3 Sample MSDP Configurations

23.3.1 MSDP and PIM-SM

Example 1

The following configuration specifies the local end, 192.0.2.1, and the remote end, 192.0.2.2, of an MSDP peering session. The statement "**assoc-msdp**" within the PIM clause associates the PIM-SM component with the MSDP component.

```
msdp on {
    peer 192.0.2.1 192.0.2.2;
};
pim on {
    sparse "sm0" {
        assoc-msdp;
        static-rp 224.0.0.0 masklen 4 192.0.2.1;
        interface fxp0 {
            boundary;
        };
        interface fxp1;
    };
};
```

Example 2

In the following configuration, peers of the MSDP peering session, 192.0.2.1 and 192.0.2.2, belong to mesh group 1. If the router 192.0.2.1 receives an SA-Advertisement message from 192.0.2.2, then it must forward the message to all other peers, 192.0.2.3 and 192.0.2.4. If the router receives an SA-Advertisement message from 192.0.2.3 or 192.0.2.4, then it must forward the message to 192.0.2.2.

```
msdp on {
    peer 192.0.2.1 192.0.2.2 mesh 1;
    peer 192.0.2.1 192.0.2.3;
    peer 192.0.2.1 192.0.2.4;
};
pim on {
    sparse "sm0" {
        assoc-msdp;
        static-rp 224.0.0.0 masklen 4 192.0.2.1;
        interface fxp0 {
            boundary;
        };
```

```
            interface fxp1 fxp2;
        };
    };
```

Example 3

In the following configuration, two peering sessions are configured. An export filter is configured such that if any (S, G) pairs received in SA messages match the filter (192.0.2.0/24, 226.1.0.0/16), then the (S, G) pairs will not be forwarded to other peers.

```
msdp on {
    peer 192.0.2.1 192.0.2.2;
    peer 192.0.2.1 192.0.2.3;
    sa-filter export 192.0.2.0 masklen 24 226.1.0.0 masklen 16;
};
pim on {
    sparse "sm0" {
        assoc-msdp;
        static-rp 224.0.0.0 masklen 4 192.0.2.1;
        interface fxp0 {
            boundary;
        };
        interface fxp1;
    };
};
```

23.3.2 MSDP Only

Example 1

In the following configuration, 192.0.2.1 is configured to peer with both 192.0.2.2 and 192.0.2.3. The PIM-SM is configured **off**; the MSDP component is not associated with the PIM-SM component.

```
msdp on {
    peer 192.0.2.1 192.0.2.2;
    peer 192.0.2.1 192.0.2.3;
};
pim off ;
```

23.4 Defaults

```
 msdp on {
     keepalive-period 75;
     peer-holdtime 90;
```

```
        sa-cache-timeout 210;
        sa-holddown 30;
        connect-retry-period 30;
    };
```

Chapter 24
Internet Group Management Protocol (IGMP)

24.1 IGMP Overview

IGMP was designed for hosts on multi-access networks to inform locally-attached routers of their multicast group memberships. Hosts inform routers of the groups of which they are members by multicasting IGMP Group Membership Reports. Once multicast routers listen for these reports, they can exchange group membership information with other multicast routers. This reporting system allows distribution trees to be formed to deliver multicast datagrams. The original version of IGMP was defined in RFC 1112, Host Extensions for IP Multicasting. Extensions to IGMP, known as IGMP version 2, include explicit Leave messages for faster pruning and are defined in RFC 2236. GateD implements only IGMP version 2, which includes interoperability with version 1 hosts. The original version of IGMP can be found at:

http://www.ietf.org/rfc/rfc1112.txt

IGMP version 2 is described in:

http://www.ietf.org/rfc/rfc2236.txt

24.2 IGMP Syntax

```
igmp ( on | off ) [ {
    [ interface interface_list [ {
        [ enable ; | disable ; ]
        [ nosend ; ]
        [ query-interval sec ; ]
        [ max-response-time sec ; ]
        [ last-mem-query-intvl sec ; ]
        [ robustness value ; ]
    } ; ]
    [ traceoptions trace_options ; ]
    [ query-interval sec ; ]
    [ max-response-time sec ; ]
    [ last-mem-query-intvl sec ; ]
    [ robustness value ; ]
} ] ;
```

More detailed descriptions of these commands can be found on page 521 of the *Command Reference Guide*.

24.3 Sample IGMP Configurations

24.3.1 Example 1: IGMP and DVMRP

This is a simple IGMP and DVMRP configuration with passive interfaces.

```
interfaces {
    interface all passive;
};
igmp yes;
dvmrp yes;
```

24.3.2 Example 2: IGMP and DVMRP

The following example enables IGMP and DVMRP on interfaces le0 and le1 only.

```
igmp yes {
    interface le0 { enable; };
    interface le1 { enable; };
};
dvmrp yes {
    interface le0 { enable; };
    interface le1 { enable; };
};
```

24.3.3 Example 3: IGMP Only

This example will enable IGMP while leaving all of the fxp interfaces with a default robustness of 2 except fxp2. The fxp2 interface will have a robustness of 3.

```
igmp yes {
    interface fxp {
        enable;
    };
    interface fxp2 {
        enable;
        robustness 3;
    };
};
```

24.3.4 Example 4: IGMP Only

The following configuration enables IGMP on all fxp interfaces. The interface fxp2 will have a robustness of 4 while all other fxp interfaces will have a robustness of 3.

```
igmp yes {
    interface fxp {
        enable;
    };
    interface fxp2 {
        enable;
        robustness 4;
    };
```

```
        robustness 3;
    };
```

24.3.5 Example 5: IGMP Only

In the following configuration, IGMP is enabled on all interfaces.

```
interfaces {
    interface all passive;
};
igmp yes;
```

24.3.6 Example 6: IGMP Only

In the following configuration, IGMP is enabled only on interfaces qe0 and qe2.

```
igmp yes {
    interface le0 { disable; };
    interface qe0 { enable; };
    interface qe1 { enable; };
    interface qe2 { disable; };
    interface qe3 { disable; };
};
```

24.3.7 Example 7: Per Interface Configuration

The following configuration illustrates how to configure interface-specific options.

```
igmp yes {
    interface all {
        enable;
        query-interval 135;
        last-mem-query-intvl 2;
        max-response-time 10;
        robustness 3;
    };
    interface fxp0 {
        enable;
        query-interval 120;
        last-mem-query-intvl 3;
     }
    interface fxp2 {
         disable;
    }
};
```

Given the configuration above, assume the router has three interfaces fxp0, fxp1, and fxp2. Interface fxp1 will have the following values:

```
query-interval 135;
last-mem-query-intvl 2;
max-response-time 10;
robustness 3;
```

Interface fxp0 will have the following values:

```
query-interval 120;
last-mem-query-intvl 3;
max-response-time 10;
robustness 3;
```

and fxp2 will be disabled.

24.4 Defaults

```
igmp yes {
    interface all {
        enable;
        query-interval 125;
        last-mem-query-intvl 1;
        max-response-time 10;
        robustness 2;
    };
    traceoptions none;
    query-interval 125;
    max-response-time 10;
    last-mem-query-intvl 1;
    robustness 2;
};
```

The configuration above is equivalent to:

```
igmp yes {
    interface all {
        enable;
        query-interval 125;
        last-mem-query-intvl 1;
        max-response-time 10;
        robustness 2;
    };
    traceoptions none;
};
```

Chapter 25
Multicast Statement

25.1 Multicast Statement Overview

The multicast statement is used to set interface-specific options such as time-to-live (TTL) thresholds and admin scope boundaries.

25.2 Multicast Statement Syntax

Scoped boundaries are configured by commands inside the **`multicast {}`** block. The syntax for the multicast block is:

```
multicast {
    interface interface_list [ threshold number ] ;
    boundary group-address
      [ ( mask mask ) | ( masklen number ) ] interface_list;
} ;
```

More detailed descriptions of these commands can be found on page 539 of the *GateD Command Reference Guide*.

25.3 Multicast Sample Configurations

```
multicast {
    interface le1 threshold 16
};
```

The example above configures interface le1 with a TTL threshold of 16.

```
multicast {
    interface le1 threshold 16
    boundary 225.255.0.0 masklen 16 le1;
};
```

The example above configures interface le1 with a TTL threshold of 16, and sets a boundary for 225.255/16 on it.

```
multicast {
    interface le1 threshold 16
};
```

The example above configures interface le1 with a TTL threshold of 16, and forces GateD to act as if downstream members of group 239.1.2.3 exist on that interface.

25.4 Multicast Statement Defaults

The defaults for multicast are on a per-interface basis. The default value for TTL is 1.

Chapter 26
Mstatic Statement

26.1 Mstatic Statement Overview

The **mstatic** statement is used to configure multicast group membership information on interfaces where it is either not desired to run IGMP, or IGMP would not be able to know of a group member. Note that it does not cause GateD to join the indicated group(s) using, for example, IGMP. Rather, it causes GateD to act as if group members are present.

26.2 Mstatic Statement Syntax

```
mstatic yes | no {
    interface interface_list {
        [ enable | disable ; ]
        [ join group-address ; ]
    } ;
} ;
```

More detailed descriptions of these commands can be found on page 543 of the *Command Reference Guide*.

26.3 Mstatic Sample Configurations

```
mstatic  yes {
    interface fxp0 {
        join 224.1.1.1 ;
    } ;
    interface fxp1 {
        join 224.1.1.1 ;
        join 225.1.1.1 ;
    } ;
};
```

The example above configures interface fxp0 with membership information for group 224.1.1.1 and interface fxp1 with membership information for groups 224.1.1.1 and 225.1.1.1.

26.4 Mstatic Statement Defaults

The default for mstatic is that no group membership is configured.

Chapter 27
Routing Information Protocol, next generation (RIPng)

27.1 RIPng Overview

Routing Information Protocol, next generation (RIPng) is an implementation of a distance-vector, or Bellman-Ford, routing protocol for local networks. RIPng is based off the RIP protocol and inherits the limitations and constraints that are in RIP.

A router running RIPng sends an update to its neighbor routers every 30 seconds. Each update contains paired values, where each pair consists of an IPv6 network address and an integer distance to that network. RIPng uses a hop count metric to measure the distance to a destination. In the RIPng metric, a router advertises directly connected networks at a metric of 1 by default. Networks that are reachable through one other gateway are 2 hops, and so on. Thus, the number of hops or hop count along a path from a given source to a given destination refers to the number of gateways that a datagram would encounter along that path. Using hop counts to calculate shortest paths does not always produce optimal results. For example, a path with a hop count 3 that crosses three Ethernets may be substantially faster than a path with a hop count 2 that crosses two slow-speed serial lines. To compensate for differences in technology, many routers advertise artificially high hop counts for slow links.

At startup, RIPng issues a request for routing information and then listens for responses to the request. If a system configured to supply RIPng hears the request, it responds with a response packet based on information in its routing database. The response packet contains destination network addresses and the routing metric for each destination.

When a RIPng response packet is received, the routing daemon takes the information and rebuilds the routing database, adding new routes and "better" (lower metric) routes to destinations already listed in the database. RIPng also deletes routes from the database if the next router to that destination reports that the route contains more than 15 hops, or if the route is deleted. All routes through a gateway are deleted if no updates are received from that gateway for a specified time period. In general, routing updates are issued every 30 seconds. In many implementations, if a gateway is not heard from for 180 seconds, all routes from that gateway are deleted from the routing database. This 180-second interval also applies to deletion of specific routes.

27.2 RIPng Syntax

```
ripng ( on | off ) [ {
        preference ripngpreference ;
        defaultmetric metric ;
        expire-time expire_time ;
```

```
        update-time update_time ;
        interface interface_list
            [ noripin ] | [ ripin ]
            [ noripout ] | [ ripout ]
            [ metricin ripngmetric ] | [ metricout ripngmetric ];
        traceoptions trace_options ;
    } ] ;
```

For more detailed information on these commands, see “Chapter 23 Routing Information Protocol, next generation (RIPng)” on page 551 of the *Command Reference Guide*.

27.3 RIPng Defaults

```
ripng on [ {
  preference 100 ;
  defaultmetric 1 ;
  update-time 30 ;
  expire-time 180 ;
  interface interface_list ripin ripout metricin ripngmetric metricout
     0 ;
  traceoptions trace_options ;
} ] ;
```

27.4 RIPng Sample Configurations

27.4.1 RIPng on All Interfaces

The following configuration turns on RIPng on all interfaces with default settings.

```
ripng yes ;
```

27.4.2 RIPng on Two Interfaces

The following configuration turns on RIPng on two specific interfaces only.

```
ripng yes {
    interface eth0 ripin ripout ;
    interface eth1 ripin ripout ;
} ;
```

Chapter 28
Route Filtering

28.1 Route Filtering Overview

Routes are filtered by specifying configuration language that will match a certain set of routes by destination, or by destination and mask. Among other places, route filters are used in **martians**, and in **import** and **export** statements.

The action taken when no match is found is dependent on the context. For instance, **import** and **export** route filters assume an **all restrict;** at the end of a list. (See "Chapter 31 Route Importation" on page 137 and "Chapter 32 Route Exportation" on page 145 for more information about **import** and **export**.)

A route will match the most specific filter that applies. Specifying more than one filter with the same destination, mask and modifiers will generate an error.

28.2 Route Filtering Syntax

Examples of all the possible formats for a route filter follow.

```
network [ ( mask mask ) | ( masklen number ) ]
    [ exact | refines | ( between lower and upper ) ]
[ inet6 | inet ] all [ exact | refines | ( between lower and upper ) ]
[ inet6 | inet ] default
host [ inet6 | inet ] host
```

More detailed descriptions of these commands can be found on page 571 in the *GateD Command Reference Guide*.

Not all of these formats are available in all places. For instance, the **host** and **default** formats are not valid for **martians**.

Certain filter contexts (such as within the BGP import and export statements) allow filters for both IPv4 (**inet**) and IPv6 (**inet6**) addresses to be defined. The keywords **inet** and **inet6** are used with the **all**, **default**, and **host** filter types to specify the address type intended. In a context that allows both address families, not specifying **inet** or **inet6** with **all** or **default** causes a filter for each address family to be defined.

28.3 Route Filtering Defaults

The default action for filter matching when no filters are present is context sensitive.

If a network filter is specified without a modifier (**exact**, **refines**, or **between**), the effective default is **exact** and **restrict**. All matching addresses with mask length greater than or equal to the configured mask are matched.

28.4 Route Filtering Examples

Example 1

The following example shows how to set up a route filter for BGP-import that allows all networks with a masklen less than 19 to pass.

```
import proto bgp autonomoussystem 12345 {
    0.0.0.0 between 0 and 18;
};
```

Example 2

If MPBGP is configured, the following imports all IPv4 and IPv6 routes from **as** 42:

```
import proto bgp as 42 {
    all ;
} ;
```

Chapter 29
Matching AS Paths

29.1 AS Path Overview

An AS path includes a list of autonomous systems that routing information has passed through to get to a specified router and an indicator of the origin of this route. The AS path is used to prevent routing loops in BGP.

This routing information can be used to prefer one path to a destination network over another. The primary method for preferring a route with GateD is to specify a list of filters to be applied to AS paths when importing and exporting routes.

Each autonomous system through which a route passes prepends its AS number to the beginning of the AS path.

AS path regular expressions are defined in RFC1164, Section 4.2. For more information about RFC1164, see:

http://www.ietf.org/rfc/rfc1164.txt

29.2 Matching AS Path Syntax

An AS path is matched using the following syntax.

```
aspath "( aspath_regexp )"
origin ( any |  igp  | egp | [ incomplete )
```

More detailed descriptions of these commands can be found on page 583 of the *Command Reference Guide*.

29.3 AS Path Regular Expressions

GateD includes a powerful implementation of AS path regular expressions. The entire AS path regular expression must be contained within parentheses. These parentheses also have meaning within the language. Left parentheses match the beginning of the AS path. Right parentheses match the end of the AS path. The alphabet (set of valid members) is the valid range of AS numbers, or more specifically, {1 ... 65535}. Also, GateD supports the following "wildcards" or expressions that can be used to build a regular expression:

. - (period) represents any valid member of the alphabet.

***** - (asterisk) matches zero or more of the preceding element/expression.

+ - (plus sign) matches one or more of the preceding element/expression.

? - (question mark) matches zero or one occurrence of the preceding element/expression.

Binary operators:

" " (AND) - any sequence of elements and/or expressions separated by a space (" ").

"**|**" (OR) - any sequence of elements and/or expressions separated by the vertical line symbol ("|").

The symbols "**[]**" are used to delimit a set of AS numbers. The set may be a list of AS numbers separated by a space or a range of AS numbers separated by a dash (**-**). If the entire list of members is prefixed with a "**^**", then the valid members are those not listed in the set. (Because a null string or empty string is not an instance in the alphabet, AS numbers such as **[^808]** will not match an empty string.) Examples of AS path regular expressions follow:

Match any single AS number as the AS path:

```
(.)
```

Match all AS paths coming from a given AS that start with 808:

```
(808 .*)
```

Match all paths that do not end with the given AS numbers but must have at least one AS:

```
(.* [^808 809])
```

Match a path that has only valid exterior AS numbers:

```
([1-64999]+)
```

Match 305 808 and exactly one other AS number other than 100:

```
(305 808 [^100])
```

Match 305 808 and any other AS number except 100 or no additional AS. That is, match either 305 808 as the complete path or 305 808 x, where x is any integer other than 100:

```
(305 808 [^100]?)
```

Match either 808 or 305 with no additional AS numbers in the path:

```
(305|808)
```

To exclude a certain AS from an arbitrary path:

```
(.* 65535 .*)
```

Chapter 30
BGP Communities

30.1 BGP Communities Overview

BGP updates carry a number of path attributes. Some of these, like the **AS_PATH**, are mandatory and appear in every update message sent. Others are optional, and may or may not appear in any given update. Of the optional attributes, two can be specified arbitrarily by administrators to ease configuration. These two attributes are "communities" and "extended communities." Both of these attributes operate by "coloring" routes received in updates where these attributes are present; every router keeps track of the set of communities and extended communities with which a route was learned. The particular communities (or extended communities) with which a route was learned can be used to indicate that a particular set of policies should be applied to those routes.

Note: EXTENDED COMMUNITIES IS AN EXPERIMENTAL AND UNSUPPORTED OPTION. GATED V.9.3.2 CURRENTLY SUPPORTS THE DRAFT-IETF-IDR-BGP-EXT-COMMUNITIES-00 VERSION OF EXTENDED COMMUNITIES.

30.2 BGP Communities and Extended Communities Configurations

The following configuration causes all routes learned from BGP peers in AS 65532 with a community of 100:140 to be installed with a GateD preference of 140, those learned with a community of 100:150 to be installed with a GateD preference of 150, and all other routes learned from BGP peers in AS 65532 to be installed with the default GateD preference for BGP (170).

```
import proto bgp as 65532 comm { comm-split 100 140; } {
    all preference 140;
};
import proto bgp as 65532 comm { comm-split 100 150; } {
    all preference 150;
};
import proto bgp as 65532 {
    all;
};
```

The following causes GateD to export to AS 65532 with the community 100:140 all routes learned from peers in AS 65533, all routes learned from peers in AS 65534 with the community 100:150, and all routes learned from peers in AS 65535 with no additional communities.

```
export proto bgp as 65532 comm-add { comm-split 100 140; } {
    proto bgp as 65533 {
```

```
            all;
        };
    };
    export proto bgp as 65532 comm-add { comm-split 100 150; } {
        proto bgp as 65534 {
            all;
        };
    };
    export proto bgp as 65532 {
        proto bgp as 65535 {
            all;
        };
    };
```

30.3 Syntax

30.3.1 BGP Communities Syntax

The syntax of **comm** is as follows:

```
comm {
    community_list ;
}
# comm-add and comm-delete are used to add and delete communities.
comm-add {
    community_list ;
}

comm-delete {
    community_list ;
}
```

More detailed descriptions of these commands can be found on page 587 of the *Command Reference Guide*.

30.3.2 BGP Extended Communities Syntax

The syntax of **ext-comm** is:

```
ext-comm {
    extended_community_list ;
}
```

The syntax of **ext-comm-add** is:

```
ext-comm-add {
    extended_community_list ;
}
```

The syntax of **ext-comm-delete** is:

```
ext-comm-delete
    extended_community_list ;
}
```

More detailed descriptions of these commands can be found on page 587 of the *Command Reference Guide*.

Chapter 31
Route Importation

31.1 Route Importation Overview

`import` statements control the importation of routes from routing protocols and the installation of the routes in GateD's Routing Information Base (RIB). The format of an **`import`** statement varies depending on the source protocol. A given import statement applies to all routes that match the specified protocol and any other item specified on the import statement, such as **`tag`**, **`as`**, **`interface`**, **`gateway`**, and so on.

31.1.1 Route Filters

All the formats allow route filters as shown below. See "Chapter 28 Route Filtering" on page 129 for a detailed explanation of how they work. When no route filtering is specified (i.e., when **`restrict`** is specified on the first line of a statement), all routes from the specified source with any specified **`as`**, **`tag`**, and so on, will match that statement. If any filters are specified, only routes that also match the specified filters will be imported. Put differently, if any filters are specified, an **`all restrict`** is assumed at the end of the list.

31.1.2 Importing Routes into Different RIBS

Normally, routes from unicast routing protocols are only imported into the unicast RIB. Routes from multicast routing protocols are only imported into the multicast RIB. However, some multicast routing protocols do not maintain their own routing table, but rely on a unicast routing protocol instead. To support these protocols, unicast routes must be imported into the multicast RIB. If the routes are not imported, only interface routes will be available to those multicast protocols.

Because MPBGP is able to tag routes to indicate to which RIBs they apply, no additional configuration is required for BGP routes.

The RIP and Redirect protocols, however, do not tag routes as being for a specific Subsequent Address Family Indicator (SAFI). Hence, GateD must be configured to import RIP or Redirect routes into the multicast RIB. See "Examples of Importation into Multicast RIBs" on page 142 to see the exact syntax of the import protocol statement. One or more RIB names may be specified (where **`multicast`** and **`unicast`** appear) as in the example below:

```
import proto rip {
    all;
    198.0.0.0 masklen 8 refines multicast unicast;
};
```

This example keeps the normal behavior of allowing all RIP routes in the unicast RIB, but also imports all routes falling under 198/8 into the multicast RIB. Additional examples are included in “Examples of Importation into Multicast RIBs” on page 142.

To import OSPF routes into the multicast RIB, you currently must import all OSPF routes as follows:

```
ospf yes {
    defaults {
    ribs unicast multicast;
...
    };
...
};
```

You cannot import OSPF routes into only the multicast RIB. Attempting to do so will be flagged as a configuration error.

31.1.3 Import Inheritance

The following parameters can be specified at multiple places within a given **import** statement:

- **fromribs**
- **toribs**
- **restrict**
- **preference**

In the case of **restrict**, placing it on an **import proto** statement restricts importation of all routes that match the **import proto** statement. Including it on a *route_filter* restricts importation of all routes that match that route filter.

In the case of **preference**, **toribs**, and **fromribs**, the most specific instances of these values are assigned to a route. Thus, any such value specified on an **import proto** statement is used only if the *route_filter* that matches a route does not specify a value. The value specified on the most specific matching *route_filter* is used. If neither the **import proto** statement nor the matching *route_filter* specifies a value, then the default value for the protocol is used.

31.2 Route Importation Syntax

```
import proto bgp
    ( as ASN ) | ( aspath aspath-regular-expression
      origin ( any | igp | egp | incomplete ) )
    [ comm { communities_list } ]
    [ ext-comm { extended_communities_list } ]
    [ preference preference ]
    [ fromribs riblist ]
    [ [ toribs ] riblist ]
```

```
    { [ route_filter
      [ restrict |
      ( [ preference preference ]
        [ fromribs riblist ]
        [ [ toribs ] riblist ] ) ] ; ]
    };
import proto bgp
    ( as ASN ) | ( aspath aspath-regular-expression
      origin ( any | igp | egp | incomplete ) )
    [ comm { communities_list } ]
    [ ext-comm { extended_communities_list } ]
    restrict;

import proto ospfase
    [ tag tagvalue ]
    [ preference preference ]
    { [ route_filter
      [ restrict |
      ( [ preference preference ]
        [ [ toribs ] riblist ] ) ] ; ]
    };
import proto ospfase
    [ tag tagvalue ] restrict ;

import proto rip
    [ tag tagvalue | interface interface_list  |  gateway gateway_list ]
    [ preference preference ] [ [ toribs ] riblist ]
    { [ route_filter
      [ restrict |
      ( [ preference preference ]
        [ [ toribs ] riblist ] ) ] ; ]
    };
import proto rip
   [ tag tagvalue | interface interface_list  |  gateway gateway_list ]
    restrict ;

import proto ripng
    [ tag tagvalue | interface interface_list  |  gateway gateway_list ]
```

```
        [ preference preference ] [ [ toribs ] riblist ]
        { [ route_filter
          [ restrict |
          ( [ preference preference ]
            [ [ toribs ] riblist ] ) ] ; ]
        };
    import proto ripng
        [ tag tagvalue | interface interface_list | gateway gateway_list ]
        restrict ;

    import proto redirect
        [ interface interface_list | gateway gateway_list ]
        [ preference preference ] [ [ toribs ] riblist ]
        { [ route_filter
          [ restrict |
          ( [ preference preference ]
            [ [ toribs ] riblist ] ) ] ; ]
        };
    import proto redirect
        [ interface interface_list | gateway gateway_list ] restrict ;
```

route_filter:

route_filter specifies a list of matching filters and the corresponding action for each filter. For more information see “Chapter 28 Route Filtering” on page 129 in *Configuring GateD*.

communities_list and *extended_communities_list*:

communities_list specifies the set of communities that are to be matched. *extended_communities_list* specifies the set of extended communities to be matched. For more information, see “Chapter 30 BGP Communities” on page 133 in *Configuring GateD*.

31.3 Route Importation Defaults

```
    import proto bgp aspath “(.*)” origin any
        { all ; } ;
    import proto ospfase
        { all ; } ;
    import proto rip
        { all ; } ;
    import proto ripng
```

```
        { inet6 all ; } ;
    import proto redirect
        { all ; } ;
```

31.4 Route Importation Examples

Example 1

The following example will import only routes from AS 203 that are stamped with community 99:

```
import proto bgp as 203
    comm {
        comm-split 203 99
    }
{
    all;
};
```

Example 2

The following example will import only routes that do not originate from AS 690:

```
import proto bgp aspath “(.* 690)” origin any {
    all restrict;
};
import proto bgp aspath “(.*)” origin any {
    all;
};
```

Example 3

The following example will import all IPv4 routes from AS 200:

```
import proto bgp as 200 {
    inet all ;
} ;
```

Example 4

The following example will import all IPv6 routes from AS 65000:

```
import proto bgp as 65000 {
    inet6 all ;
} ;
```

Example 5

The following example will import all IPv6 and IPv4 routes from AS 65000:

```
import proto bgp as 65000 {
    all ;
```

```
} ;
```

Example 6

The following example will import all RIP routes:

```
import proto rip {
    all ;
} ;
```

Example 7

The following example will import RIP routes based on tags:

```
import proto rip tag 230 {
    all ;
} ;
```

31.4.1 Examples of Importation into Multicast RIBs

Example 1

Example 1 keeps the normal behavior of allowing all RIP routes in the unicast RIB, but also imports all routes falling under 198/8 into the multicast RIB.

```
import proto rip {
    all;
    198.0.0.0 masklen 8 refines multicast unicast;
};
```

Example 2

Example 2 imports all of the RIP routes into the multicast RIB (as well as the usual unicast RIB).

```
import proto rip {
    all multicast unicast;
};
```

Example 3

Example 3 imports all of the RIP routes into the unicast and multicast RIBs.

```
import proto rip unicast multicast {
    all ;
};
```

Example 4

Example 4 inserts all of the OSPF routes into the unicast and multicast RIBs.

```
ospf yes {
    defaults {
        ribs unicast multicast;
...
    };
...
};
```

Chapter 32
Route Exportation

32.1 Route Exportation Overview

The **import** statement controls which routes that are received from other systems are used by GateD, and the **export** statement controls which routes are advertised by GateD to other systems. Like the **import** statement, the syntax of the **export** statement varies slightly per protocol. The syntax of the **export** statement is similar to the syntax of the **import** statement, and the meanings of many of the parameters are identical. The main difference between the two is that while route importation is just controlled by source information, route exportation is controlled by both destination and source.

The outer portion of a given **export** statement, the **export** target statement, specifies the destination of the routing information you are controlling. The middle portion, the **export** source statement, restricts the sources of importation that you wish to consider. The innermost portion is a route filter used to select particular routes.

The **export** command specifies the policy for dynamically advertising routes from GateD to other systems. Any number of **export** commands can be given, provided that each **export** target statement is unique.

The **export** target statement specifies by which protocol the route will be advertised and how it will be advertised. It can limit to which neighbors of a given protocol the routes are sent and modify a route's attributes before it is sent. Obviously, the route attributes that exist or can be set are dependant on the outgoing protocol. (For example, you cannot set an AS Path for an OSPF route.) Only one **export** target statement can be given per **export** command.

The union of all **export** source statements represents a filter that determines the routes that will be exported. The filter will match in order and is succeeded by an implied restriction, so anything not matched by one of the **export** source statements will not be advertised. As a result, the following two statements are identical:

```
export proto rip restrict;
export proto rip { };
```

The same statements about order and restriction apply to multiple instances of the **export** command as a whole. So, if no **export** commands are given, then default behavior will ensue; however, if even one **export** statement is given, then no other exporting will occur.

32.1.1 Export Inheritance

The following parameters can be specified at multiple places within a given **export** statement:

- **restrict**
- **metric**

In the case of **restrict**, placing it on an **export target** statement restricts exportation of all routes matching the **export target** statement. Including it on an **export source** statement restricts exportation of all routes matching the **export source** statement. Including it on a *route_filter* restricts exportation of all routes that match that route filter.

In the case of export **metric**, the most specific instance of a **metric** value is assigned to a route. Thus, any **metric** value specified on an **export target** statement is used only if neither the **export source** statement nor the *route_filter* that matches a route specifies a value. The value of the most specific matching *route_filter* is used. If neither the **export target** statement, the matching **export source** statement, nor the matching *route_filter* specifies a metric value, then the default metric value for the protocol is used.

32.2 Export Syntax

export target statements are one or more of the following:

```
export proto bgp as autonomous_system restrict ;
export proto bgp as autonomous_system
    [ comm-add { communities_list } ]
    [ comm-delete { communities_list } ]
    [ ext-comm-add { extended_communities_list } ]
    [ ext-comm-del { extended_communities_list } ]
    [ metric metric ] {
        export_source_statements
    } ;

export proto rip
    [ interface interface_list | gateway gateway_list ]
    restrict ;
export proto rip
    [ tag tagvalue ]
    [ interface interface_list | gateway gateway_list ]
    [ metric metric ] {
        export_source_statements
    };

export proto ripng
    [ interface interface_list | gateway gateway_list ]
    restrict ;
export proto ripng
```

```
    [ tag tagvalue ]
    [ interface interface_list | gateway gateway_list ]
    [ metric metric ] {
        export_source_statements
    };

export proto ospfase restrict ;
export proto ospfase [ type type ] [ tag tagvalue ] [ metric metric ] {
    export_source_statements
};

export proto ospfnssa restrict ;
export proto ospfnssa [ type type ] [ tag tagvalue ] [ metric metric ] {
    export_source_statements
};

export proto isis restrict ;
export proto isis [ metric-type type ] [ level level ] [ metric metric ] {
    export_source_statements
} ;
```

export_source_statements are one or more of the following:

```
    proto bgp ( as autonomous_system ) | ( aspath aspath_regular_expression
        origin ( any | igp | egp | incomplete ) )
        [ comm communities_list ]
        [ ext-comm extended_communities_list ]
        restrict;
    proto bgp ( as autonomous_system ) | ( aspath aspath_regular_expression
        origin ( any | igp | egp | incomplete ) )
        [ comm communities_list ]
        [ ext-comm extended_communities_list ]
        [ noagg ] [ metric metric ]
        { [ route_filter
          [ fromribs riblist ] [ restrict | ( metric metric ) ] ; ] } ;

    proto rip
        [ tag tagvalue | interface interface_list | gateway gateway_list ]
```

```
    restrict ;
proto rip
    [ interface interface_list | gateway gateway_list | tag tagvalue ]
    [ metric metric ] [ noagg ]
    { [ route_filter
      [ fromribs riblist ] [ restrict | ( metric metric ) ] ; ] } ;

proto ripng
    [ tag tagvalue | interface interface_list | gateway gateway_list ]
    restrict ;
proto ripng
    [ interface interface_list | gateway gateway_list | tag tagvalue ]
    [ metric metric ] [ noagg ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;

proto ospf [ type type ] [ tag tag ] restrict ;
proto ospf [ type type ] [ tag tag ] [ metric metric ] [ noagg ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;

proto ospfase [ type type ] [ tag tag ] restrict ;
proto ospfase [ type type ] [ tag tag ] [ metric metric ] [ noagg ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;

proto direct [ ( interface interface_list ) ] restrict ;
proto direct [ ( interface interface_list ) ]
    [ noagg ] [ metric metric ]
    { [ route_filter
      [ fromribs riblist ] [ restrict | ( metric metric ) ] ; ] } ;

proto static [ interface interface_list ] restrict ;
proto static [ interface interface_list ]
    [ metric metric ] [ noagg ]
    { [ route_filter
      [ fromribs riblist ] [ restrict | ( metric metric ) ] ; ] } ;

proto kernel [ interface interface_list ] restrict ;
proto kernel [ interface interface_list ] [ metric metric ] [ noagg ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;
```

```
proto isis [ internal | external ] restrict ;
proto isis [ internal | external ]
    [ noagg ] [ metric metric ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;

proto aggregate restrict ;
proto aggregate [ noagg ] [ metric metric ]
    { [ route_filter [ restrict | ( metric metric ) ] ; ] } ;
```

32.3 Export Defaults

```
export proto rip {
    proto rip {
        all;
    };
    proto direct {
        all;
    };
};
export proto ripng {
    proto direct {
        all;
    };
};
```

32.4 Export Examples

32.4.1 Exporting to BGP

Example 1

This example configures BGP to export all active BGP routes learned from AS 202 and three static routes and any of their static subnets to BGP peers in AS 201.

Note: The default policy of exporting all 'direct' routes is no longer in effect if export policy is given.

```
export proto bgp autonomoussystem 201 {
    proto bgp autonomoussystem 202 {
        all;
    };
    proto static {
```

```
            223.1.0/24;
            223.2.0/24;
            223.3.0/24;
        };
    };
```

Example 2

This example exports all IPv4 and IPv6 static routes and all routes learned from AS 202, to AS 201 (assuming MPBGP is enabled).

```
    export proto bgp as 201 {
        proto bgp as 202 {
            all;
        };
        proto static {
            all ;
        } ;
    } ;
```

32.4.2 Exporting to RIP

This example configures all static and direct routes to be exported into the RIP protocol with tag 2112.

Note: The default policy of exporting all 'direct' routes is no longer in effect if export policy is given.

```
    export proto rip tag 2112 {
        proto static {
            all;
        };
        proto direct {
            all;
        };
    };
```

32.4.3 Exporting to RIPng

This example configures all static and direct IPv6 routes to be exported into the RIPng protocol with tag 2112.

Note: The default policy of exporting all 'direct' routes is no longer in effect if export policy is given.

```
    export proto ripng tag 2112 {
        proto static {
            all;
```

```
        };
        proto direct {
            all;
        };
    };
```

32.4.4 Exporting to OSPF ASE and NSSA

Example 1

This example configures all static routes to be exported into OSPF ASE (Type-5 LSAs).

```
    export proto ospfase {
        proto static {
            all;
        };
    };
```

Example 2

A similar configuration can be used to export all static routes into OSPF NSSA (Type-7 LSAs).

```
    export proto ospfnssa {
        proto static {
            all;
        };
    };
```

32.4.5 Exporting to ISIS

This example configures all IPv4 and IPv6 static routes to be exported into ISIS external reachability. The routes shall be originated in Level 2 LSPs only.

```
    export proto isis level 2 {
        proto static {
            all;
        };
    };
```

32.4.6 Exporting RIP Routes

This example configures certain RIP routes to be exported into OSPF ASE using a metric of 1 in the ASE LSAs. The 192.168.10/24 route shall be exported using a metric of 2.

```
    export proto ospfase metric 1 {
        proto rip {
            192.168.10/24 metric 2;
```

```
            192.168.11/24 restrict;
            192.168.12/24;
        };
    };
```

32.4.7 Exporting OSPF Routes

This example configures all OSPF routes to be exported into RIP.

Note: The OSPF routes that will match are OSPF internal routes, not ASE-based routes.

```
export proto rip {
    proto ospf {
        all;
    };
};
```

32.4.8 Exporting Routes from Non-routing Protocols

The 'aggregate' protocol is not a routing protocol but a classification of routes configured by the user. The following example exports all aggregate routes (see the aggregate clause) into OSPF ASE.

```
export proto ospfase {
    proto aggregate {
        all;
    };
};
```

32.4.9 Exporting by AS Path

This example uses a simple AS Path regular expression to export routes with an AS Path of 204, 203 into RIP, placing the tag 2112 in the RIP tag field.

```
export proto rip tag 2112 {
    proto bgp aspath “(204 203)” origin any {
        all;
    };
};
```

32.4.10 Exporting by Route Tag

Where supported by the routing protocol, export policy may use route tags. The following example exports all RIP routes with tag 2112 into OSPF ASE.

```
export proto ospfase {
    proto rip tag 2112 {
        all;
    };
```

```
};
```

Chapter 33
Route Aggregation and Generation

33.1 Route Aggregation Overview

Route aggregation is a method of generating a more general summary route, given the presence of a more specific route. It is used, for example, at an autonomous system border to generate a route to a network to be advertised via BGP, given the presence of one or more subnets of that network learned via RIP. No aggregation is performed unless explicitly requested in an **`aggregate`** statement.

Route aggregation is also used by regional and national networks to reduce the amount of routing information passed. With careful allocation of network addresses to clients, regional networks can announce one route to regional networks instead of hundreds.

Aggregate routes are not actually used for packet forwarding by the originator of the aggregate route, only by the receiver (if it wishes). A router is supposed to respond with an ICMP network unreachable message if the router receives a packet that does not match one of the component routes that led to the generation of an aggregate route. This message is to prevent packets for unknown component routes from following a default route into another network, where they would be forwarded back to the border router, and around and around again and again, until their TTL expired. Sending an unreachable message for a missing piece of an aggregate is only possible on systems with support for reject routes.

A slight variation of aggregation is the generation of a route based on the existence of certain conditions. This is sometimes known as the "route of last resort." This route inherits the next hops and AS path from the contributor specified with the lowest (most favorable) preference. The most common usage for this is to generate a default based on the presence of a route from a peer on a neighboring backbone.

33.1.1 Aggregate and Generate Inheritance

The following parameters can be specified at multiple places within a given **`aggregate`** or **`generate`** statement:

- **`restrict`**
- **`preference`**

In the case of **`restrict`**, placing it on an **`aggregate`**/**`generate`** source statement restricts inclusion of all routes from the specified protocol in the aggregate. Including it on a *`route_filter`* excludes all routes that match that route filter.

In the case of aggregation preference, the most specific instance of a **`preference`** value is assigned to a route. Thus, any **`preference`** value specified on an **`aggregate`**/**`generate`** statement is used only if neither the **`aggregate`**/**`generate`** source statement nor the *`route_filter`* that matches a route specifies a value. The value of the most specific

matching *route_filter* is used. If neither the **aggregate**/**generate** statement, the matching **aggregate**/**generate** source statement, nor the matching *route_filter* specifies a preference value, then the default aggregate preference value of 130 is used.

33.2 Aggregation and Generation Syntax

```
aggregate ( [ inet6 ] default |
  network ( mask mask | masklen masklen | / masklen ) )
    [ preference preference_value ]
    [ bgp ]
    [ brief ]
    [ [ toribs ] ( [ unicast | multicast | unicast multicast ) ]
    {
        aggregate_list
    };
generate ( [ inet6 ] default |
  network ( mask mask | masklen masklen | / masklen ) )
    [ preference preference_value ]
    [ [ toribs ] ( unicast | multicast | unicast multicast ) ]
    [ noinstall ]
    {
        aggregate_list
    };
```

aggregate_list:

Aggregate lists are composed of one or more of the following:

```
    proto bgp
        [ ( as ASN ) | ( aspath aspath-regular-expression
        origin ( any | igp | egp | incomplete ) ) ]
        [ comm { communities_list } ]
        [ ext-comm { extended_communities_list } ]
        [ preference preference ]
        { [ route_filter [ preference preference | restrict ] ; ] } ;
    proto bgp
        [ ( as ASN ) | ( aspath aspath-regular-expression
        origin ( any | igp | egp | incomplete ) ) ]
        restrict  ;

    proto rip
        [ tag tagvalue ]
```

```
    restrict ;
proto rip
    [ tag tagvalue ]
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ripng
    [ tag tagvalue ]
    restrict ;
proto ripng
    [ tag tagvalue ]
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ospf [ tag tagvalue ] restrict ;
proto ospf [ tag tagvalue ] [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ospfase [ tag tagvalue ] restrict ;
proto ospfase [ tag tagvalue ] [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto direct restrict ;
proto direct
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto static restrict ;
proto static
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto kernel restrict ;
proto kernel
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto isis restrict ;
```

```
proto isis
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto aggregate restrict ;
proto aggregate
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto all restrict ;
proto all
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;
```

More detailed descriptions of these commands can be found on page 667 of the *Command Reference Guide*.

Routes that match the route filters are called “contributing” routes. They are ordered on the list of contributing routes for a given aggregate route according to the aggregation preference value that applies to them, with the lowest (best) preference values coming first. The aggregation preference value can be specified on the **aggregate** or **generate** statement, any **aggregate** source statement, or any *route_filters*. In addition to being used to order contributing routes, the aggregation preference value associated with the first contributor on this ordered list is used as the route preference value for the aggregate route itself.

When ordering contributing routes on the contributor list, if two contributors have the same aggregation preference value, then the contributor with the lowest route preference is ordered before the other contributors with the same aggregation preference value.

In the case of **generate**, in addition to using the aggregation preference value of the first contributor in the list as the aggregate route’s preference, the nexthops used for the aggregate are taken to be those of the first contributor in the list.

The aspath for an aggregate is formed by combining the aspath of each contributor. If BGP rules are configured for aggregation, the contributor order can impact the MED and nexthop values in the combined aspath generated for the aggregate. This is due to the fact that the values of MED and nexthop from the first valid BGP contributor route encountered while traversing the contributor list in order are used in the combined aspath.

A route can contribute only to an aggregate route that is more general (less specific) than itself; it must match the aggregate under its mask. Any given route can contribute only to one aggregate route, which will be the most specific configured, but an aggregate route may contribute to a more general aggregate.

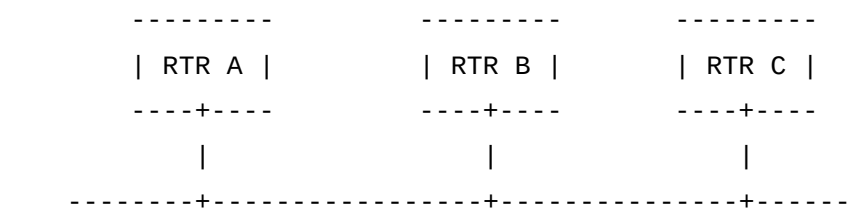
Note that a route is only considered as a potential contributor to aggregate routes for which it is a more specific route. Therefore, a filter of **all** only pertains to more specific routes of the aggregate. Currently, specifying a *network* type filter for a network that is

not a more specific instance of the aggregate to which it pertains is not treated as an error even though it will never match any contributors.

33.3 Exporting Generated vs. Aggregated Routes

If you create an aggregate and export it, the result achieved is that which is expected: for the general aggregate, a loopback (reject) route is installed in the kernel, and the route is advertised with the aggregating router as the next hop.

Consider the following topology



wherein routers A and B are OSPF peers and routers B and C are BGP peers. Let router A advertise a static route to 223.50. Let router B have the following configuration

```
aggregate 223 masklen 8 {
    proto ospfase {
         223 masklen 8 refines;
    };
}

export bgp peeras 123 {
    proto aggregate {
        all;
    };
};
```

Router B will install in its kernel

```
223.50 gw RTR A
223/8 gw 127.0.0.1 reject
```

and will advertise to Router C a route to 223/8 gw Router B. On the other hand, if the same configuration is used but with **generate** instead of **aggregate**, so we have

```
generate 223 masklen 8 {
    proto ospfase {
        223 masklen 8 refines ;
    };
};
```

then two things happen. Router B, instead of installing a reject route for portions of the aggregate that do not have specific matches, will instead install the following routes in the kernel:

```
223/8 gw RTR A
223.50 gw RTR A
```

The second difference is that the aggregate route advertised to router C will be 223/8 gw router A.

You can also configure an aggregate 10/7 on behalf of another client AS (for example, AS 65003):

```
aggregate 10.0.0.0 masklen 7 {
    proto bgp as 65003{
        10.0.0.0 masklen 8;
        11.0.0.0 masklen 8;
    };
};
```

33.4 Aggregating into Unicast and Multicast RIBs

RIBs need not be specified for aggregate routes. By default, an aggregate applies to all RIBs to which any contributing route applies. For example, an aggregate applies to the Unicast RIB if and only if any contributing route applies to the Unicast RIB.

Examples

```
aggregate 10.0.0.0 masklen 8 {
    proto static {
        10.0.0.0 masklen 8 refines;
      };
};
```

If any static route in the Unicast RIB matches the route filter, the aggregate will exist in the Unicast RIB. Likewise, for the Multicast RIB.

RIB limits can, however, be specified. By default, the limit is all RIBs (i.e., all RIBs to which any contributing route applies). This default can be overridden with a more specific limit, as in the example below:

```
aggregate 10.0.0.0 masklen 8 unicast {
    proto static {
        10.0.0.0 masklen 8 refines;
    };
};
```

The above aggregate applies only to the Unicast RIB (and only if a contributing route is in the Unicast RIB). Contributing routes in other RIBs are ignored.

Chapter 34
Route Flap Damping

34.1 Route Flap Damping Overview

GateD can be configured to suppress propagation of unstable BGP routes. This feature is commonly referred to as “route flap damping”. For each route to a destination from each peer, GateD maintains an instability metric. Whenever the peer deletes or changes its route to the destination, GateD increments the associated instability metric. The metric decays exponentially with time, with a configurable half-life time; the decay rates can be configured differently when the destination is reachable or unreachable.

When a route’s instability metric crosses a specified upper threshold, GateD suppresses the route. GateD will reuse the route only when the instability metric goes below another configurable lower threshold. GateD suppresses usage of routes that have a stability history that crosses a given configurable threshold.

34.2 Route Flap Damping Syntax

The syntax for the **dampen-flap** clause is as follows:

```
dampen-flap {
   [ suppress-above flap-metric ; ]
   [ reuse-below flap-metric ; ]
   [ max-flap flap-metric ; ]
   [ reach-decay time ; ]
   [ unreach-decay time ; ]
   [ keep-history time ; ]
};
```

The **dampen-flap** statement follows the **bgp** statement and precedes the policy statements in the run-time configuration file. If the **dampen-flap** statement is absent, GateD will not maintain a route instability history. If a **dampen-flap** statement is present, but without any parameters, the default value of the parameters is used. If a reconfiguration changes the values of any parameter, GateD erases all previous route instability history.

More detailed descriptions of these commands can be found on page 701 of the *Command Reference Guide*.

Chapter 35
SNMP Multiplexing (SMUX)

35.1 SMUX Overview

When the **smux** clause is used, GateD will attempt to contact an SNMP master-agent on the local host via the SMUX (SNMP Multiplexing) protocol over TCP. The SMUX protocol is described in RFC 1227.

Version 9.2 of GateD supports SMUX as the only way to interact with the MIB modules. Only SNMP version 1 is supported, and all MIB variables are read-only. Prior versions of GateD included an embedded standalone SNMP agent; this was removed in the 9.0 code base.

Upon contacting the master agent, a string password and SNMP Object Identifier identity are passed for authentication purposes. If the authentication succeeds, GateD will register the routing MIB subtrees and request that it be contacted when the master agent receives queries for these subtrees. When the master agent receives such a query from a management station, it will be passed to GateD.

The GET and GETNEXT operations are both supported.

35.2 SMUX Syntax

```
smux ( on | off )
[ {
    traceoptions smuxtraceoptions ;
    port smuxport ;
    password string ;
} ] ;
```

More detailed descriptions of these commands can be found on page 709 of the *Command Reference Guide*.

35.3 SNMP Query Examples

GateD uses a hard-coded identity of .1.3.6.1.4.1.4.3.1.4. After GateD has been started and has connected to a master agent, it will respond to queries that fall within its supported MIBs. For example, using the ucd-snmp **snmpget** utility, the following command would retrieve the variable *bgp.bgpLocalAs*, if both GateD and the master agent were running on the local machine, using version-1 SNMP.

```
snmpget -v 1 localhost public 15.2.0
```

This sends an SNMP GET to the local master agent, which in turn queries GateD for the value. For SNMP tables, the **snmpwalk** utility can be used to walk an entire table, using GET-NEXTs. Here is an example command to walk the entire BGP Peer Table for the peer address 192.168.10.1:

```
snmpwalk -v 1 localhost public 15.3.1.1.192.168.10.1
```

35.4 SMUX Sample Configurations

35.4.1 Simple Configuration

GateD is known to interoperate with the ucd-snmp "snmpd" agent. The following is a simple configuration that allows snmpd to pass SNMP queries to gated via SMUX.

```
smux on {
    traceoptions "/tmp/smuxlog" all ;
    password "gated";
};
```

35.4.2 Simple Configuration with Port

The master agent may be listening for SMUX connections on a different port than the registered port of 199. It is possible to configure GateD to attempt the connection on a user-specified port.

```
smux on {
    traceoptions "/tmp/smuxlog" all;
    port 2112 ;
};
```

Chapter 36
Frequently Asked Questions

36.1 Kernel Interactions

1. Does GateD support dynamic configuration on the fly? Does GateD support command line coding?

GateD uses a flat file, rather than a command line interface. To re-read the file after configuration changes, GateD must receive a SIGHUP. This can be accomplished either directly through the UNIX signaling mechanism or by use of the GDC (GateD Controller). However, this configuration is "dynamic," in that states are retained between reconfigs whenever possible. For example, BGP connections are not terminated.

2. Does GateD use the UNIX mbuf structure or its own buffer?

GateD does not use UNIX mbuf structures. GateD uses its own buffers, dynamically allocated from process address space.

3. Does GateD require pre-emptive (such as UNIX, Windows, or Linux) or non-preemptive operating systems (such as PSOS and any real-time embedded systems)?

A complete listing of currently supported operating systems is available. GateD runs in a single process with co-operative multitasking inside the processes. We do not require preemptive or non-preemptive operating systems as long as the process receives adequate cycles.

4. Does GateD require a UNIX-style "fork" function?

The "**`dump`**" feature in GateD uses **`fork(2)`** to spawn a child process. The child dumps internal state to a file and exits. If this feature is not necessary, it may be removed without affecting operation.

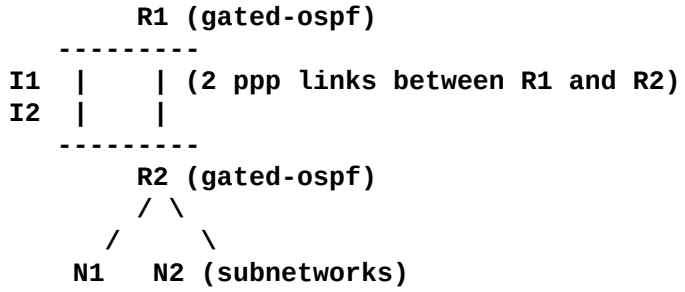
5. Does GateD require timer functions from the operating system (for example, 1/10 or 1/100 sec time granularity, recursive timer, sleep, signal, and so on)?

In releases that support signal I/O and interval timer, the **SIGALARM** and **SIGIO** signals may be sent by the operating system to allow GateD to respond to certain critical events, for example, expiration of an OSPF **HELLO** timer. With normal timers, a timer granularity of 1/10 second is sufficient for operation.

36.2 Protocols

36.2.1 OSPF

1. In a network with the following topology:



there are two point-to-point links, I1 and I2, between routers R1 and R2. R2 is connected to networks N1 and N2. Both routers are running gated-ospf. How can I route the traffic from R1 to R2 on I1 only if it is destined to N1, and on I2 only if it is destined to N2?

Put I2 and N2 in one area and I1 and N1 in another area. OSPF prefers intra-area routes over inter-area. An example configuration would be:

For R1:

```
ospf yes {
    priority 1;
    backbone {
        interface I1;
    };
    area 0.0.0.1 {
        interface I2;
    };
};
```

For R2:

```
ospf yes {
```

```
        priority 1;
        backbone {
            interface I1;
            interface N1;
        };
        area 0.0.0.1 {
            interface I2;
            interface N2;
        };
    };
```

2. The following message appears when GateD is restarted. "gated[28100]: task_get_proto: getprotobyname("ospf") failed, using proto 89" What does this message mean?

The `/etc/protocol` file doesn't contain an ospf entry. The entry should look something like this:

```
ospf    89  OSPFIGP     # Open Shortest Path First IGP
```

The 89 is the assigned Internet protocol number specified in RFC 1700. GateD will use the default value and continue if possible.

3. OSPF does not seem to install multiple equal-cost routes to a destination. I know OSPF supports ECMP (Equal-Cost Multipath Routing). What's wrong?

The OSPF module supports ECMP by installing more than one next hop in the RIB. Kernel support for this feature, however, is not widely available. In order for all of the equal-cost routes to a destination to be installed, the krt_rt_xxx module (and kernel) must support this.

36.2.2 BGP

1. How can I increase the number of BGP peers GateD will allow?

Specify a value for **`RTBIT_SIZE`** in your config file and recompile GateD. Each increment of **`RTBIT_SIZE`** provides 32 additional bits for 32 additional peers.

Example: **`options RTBIT_SIZE=4`** will allow up to 128 peers. The default value of **`RTBIT_SIZE`** is 1.

2. Why is GateD changing the NEXT_HOP attribute when advertising a route to an internal peer?

The BGP RFC states: “When a BGP speaker advertises the route to another BGP speaker located in its own autonomous system, the advertising speaker shall not modify the NEXT_HOP attribute associated with the route.” See “Chapter 14 Border Gateway Protocol (BGP)” on page 61 for more information about the BGP statement.

Basically, GateD is designed not to modify the NEXT_HOP if it believes that its IBGP peer will be able to figure out how to reach the address it depicts. GateD will go ahead and rewrite the NEXT_HOP if it believes that the peer will not know how to reach the depicted address.

In the case of group type internal, GateD knows (because you have so configured it) that its peers do not do BGP-IGP next hop resolution. GateD also knows that all of its peers are L2-adjacent, so it rewrites the NEXT_HOP to something it knows its peer will be able to reach at L2.

If you wish to make GateD conform to the RFC instead of allowing this behavior, you can use group type routing with **`interface all`** specified. In the case of **`group type routing ... interface all`**, GateD knows that (a) its peers are resolving BGP-IGP (this is a property of group type routing) and (b) NEXT_HOPs via any interface are known via the IGP (this is what "**`interface all`**" means). So it won't rewrite any NEXT_HOPs.

3. I keep seeing error messages about an unsupported optional parameter when trying to peer with a Cisco®. What is the problem?

Some versions of Cisco® IOS have a capabilities negotiation bug. The intended behavior of capabilities negotiation is to resend a BGP open message without the optional parameter once it receives a notification from GateD stating that the parameter is unsupported. You should upgrade your Cisco® or apply this workaround: **`neighbor x.x.x.x dont-capability-negotiate`**. For more information about capabilities negotiation, refer to **`draft-ietf-idr-bgp4-cap-neg-03.txt`**.

4. How can I configure a peer that is not on the same network?

Use the **`gateway`** keyword on the peer statement:

```
group type external peeras 65000 {
    peer a.b.c.d gateway w.x.y.z;
};
```

where *`a.b.c.d`* is your peer's IP address and *`w.x.y.z`* is the next hop that GateD should use to find *`a.b.c.d`*.

5. Why isn't BGP advertising my static routes?

If no export policy is specified, BGP will advertise only direct (interface) routes. To export static routes, you will need an export statement like this:

```
export proto bgp as 65500 {
    proto static {
```

```
            all;
        };
    };
```

Note that once export policy has been defined for BGP, GateD needs to be explicitly configured in order to export direct (interface) routes. Use proto direct to do this. See “Exporting to BGP” on page 149 for more information on configuring BGP export policy.

6. Why is GateD ignoring my MEDs?

If you want GateD to pay attention to incoming metrics, you need to specify the **med** keyword on the group statement:

```
group type external peeras 65530 med {
    peer 192.168.10.2;
};
```

The default behavior is for GateD to ignore incoming metrics.

7. What is BGP's default import and export behavior?

GateD will import all routes from a configured peer unless otherwise configured. If no export policy is specified, BGP will advertise only direct (interface) routes. See “Examples of Importation into Multicast RIBs” on page 142 and “Exporting to BGP” on page 149 for more information on configuring BGP policy.

8. If, on a Cisco® router, a route is redistributed (exported) from another protocol, such as static or OSPF, into BGP, what should the origin of the route be?

If you redistribute using the "network x.x.x.x" command, your Cisco® router will automatically set the origin to "IGP." If you use "redistribute," your Cisco® router will use origin incomplete. Either will work. A description of the origin path attribute follows.

ORIGIN (Type Code 1):

ORIGIN is a well-known mandatory attribute that defines the origin of the path information. The data octet can assume the following values:

Value	Meaning
0	IGP - Network Layer Reachability Information is interior to the originating AS.
1	EGP - Network Layer Reachability Information is learned via EGP.

Value	Meaning
2	INCOMPLETE - Network Layer Reachability Information is learned by some other means.

Chapter 37
Glossary of Terms

Terms used in descriptions throughout this document are defined below.

adjacency

Adjacency is a relationship formed between selected neighboring routers for the purpose of exchanging routing information. Not every pair of neighboring routers becomes adjacent.

autonomous system

An autonomous system is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the autonomous system, and using an exterior gateway protocol to route packets to other autonomous systems. Since this classic definition was developed, it has become common for a single autonomous system to use several interior gateway protocols and sometimes several sets of metrics within an autonomous system. The use of the term “autonomous system” stresses that even when multiple IGPs and metrics are used, the administration of an autonomous system appears to other autonomous systems to have a single coherent interior routing plan and to present a consistent picture of what networks are reachable through it. The autonomous system is represented by a number between 1 and 64511, assigned by the Regional Internet Registries (RIR) which has been delegated this responsibility by the Internet Assigned Numbers Authority (IANA).

BGP - Border Gateway Protocol

The Border Gateway Protocol (BGP) is an exterior (inter-domain) routing protocol used for exchanging routing information between autonomous systems. BGP routes contain "path attributes", which provide various information about reachable network destinations. These attributes contain loop-prevention information, information about route origin, and other properties that may be administratively set on the routes to aid in BGP route selection.

BGP is described in more detail in the BGP Protocol section. See “Chapter 14 Border Gateway Protocol (BGP)” on page 61.

designated router

A designated router in OSPF is a router that generates a link-state advertisement for the multiaccess network and assists in running the protocol. Each multiaccess network that has at least two attached routers has a designated router.

destination

A destination is any network or any host.

DVMRP - Distance Vector Multicast Routing Protocol

DVMRP is the original IP multicast routing protocol. DVMRP was designed to run over multicast capable LANs (like Ethernet) as well as through non-multicast capable routers. In the

case of non-multicast capable routers, the IP multicast packets are “tunneled” through the routers as unicast packets.

EGP - Exterior Gateway Protocol

EGP can mean one of two things. First, it can refer generically to the class of routing protocols for inter-domain routing - an exterior gateway protocol.

Second, it can refer specifically to the EGP, the historic predecessor to BGP.

forwarding table

The forwarding table is the table in the kernel that controls the forwarding of packets. The forwarding table is also known in OSI terms as a “forwarding information base,” or FIB. The forwarding table contains tuples that are used to determine how packets are forwarded. These tuples consist of the following fields:

- network (or CIDR prefix)
- next hop(s)
- the interface that the packet goes out

The table that GateD uses internally to store routing information that it learns from routing protocols is a “routing table,” known in OSI terms as a “routing information base,” or “RIB.”

gateway

A gateway is an intermediate destination by which packets are delivered to their ultimate destination. A gateway is a host address. Gateways are specified in the format of their address family, which in IP is the dotted-quad format, for example, 192.0.2.1. In IP, it may also be specified as an 8-digit hexadecimal string preceded by 0x, for example, 0xc0000201.

If "**`options noresolve`**" is not specified, a gateway can be a symbolic hostname, for example, gw.example.com. The numeric forms are much preferred over the symbolic form.

gateway_list

A *gateway_list* is a list of one or more gateways separated by white space.

IGMP - Internet Group Management Protocol

IGMP was primarily designed for hosts on multi-access networks to inform locally-attached routers of their group membership information. Hosts inform routers by multicasting IGMP Host Membership Reports. Once multicast routers listen for these reports, they can exchange group membership information with other multicast routers. This reporting system allows distribution trees to be formed to deliver multicast datagrams.

IGP - Interior Gateway Protocol

IGP is one of a class of routing protocols used to exchange routing information within an autonomous system. (A detailed explanation of interior gateway protocols is available in “Chapter 24 Internet Group Management Protocol (IGMP)” on page 119.)

inter-domain routing

Inter-domain routing protocols are used to exchange routing information between autonomous systems. (See also “EGP - Exterior Gateway Protocol” on page 174.)

interface

The *interface* is the host address of an attached interface. This is the address of a broadcast, NBMA or loopback interface, and the remote address of a point-to-point interface. As with any host address, it may be specified symbolically.

interface

The interface is the connection between a router and one of its attached networks. A physical interface may be specified by a single IP address, domain name, or interface name (unless the network is an unnumbered point-to-point network). Multiple levels of reference in the configuration language allow identification of interfaces using wildcard, interface type name, or delete word address. Be careful with the use of interface names, because there may be more than one address per interface. Dynamic interfaces can be added or deleted, and indicated as up or down, as well as changed to address, netmask and metric parameters.

interface_list

An *interface_list* is a list of one or more interface names, including wildcard names (names without a number) and names that may specify more than one interface or address, or the token "all" for all interfaces. See "Chapter 7 Interface Statement" on page 23 for more information.

intra-domain routing

Intra-domain routing protocols are used to exchange reachability information within an autonomous system (AS). (See also "IGP - Interior Gateway Protocol" on page 174.)

IS-IS - Intermediate System to Intermediate System

Intermediate System to Intermediate System (IS-IS) is one of a class of interior gateway protocols (See also "IGP - Interior Gateway Protocol" on page 174.) IS-IS is a link-state interior gateway protocol originally developed for routing ISO/CLNP (International Organization for Standardization/Connectionless Network Protocol) packets. IS-IS is described in more detail in "Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55. The version distributed in GateD routes IP.

local_address

The *local_address* is the host address of an attached interface. This is the address of a broadcast, NBMA, or loopback interface, and the local address of a point-to-point interface. As with any host address, it may be specified symbolically.

mask

A *mask* is a means of subdividing networks using address modification. A mask is a dotted quad specifying which bits of the destination are significant. (Except when used in a route filter (see "route filter" on page 177), GateD supports only contiguous masks.)

mask length

The mask length is the number of contiguous one bits at the beginning of the mask.

metric

A metric is one of the units used to help a system determine the best route. Metrics may be based on hop count, routing delay, or an arbitrary value set by the administrator, depending on the type of routing protocol. Routing metrics may influence the value of assigned internal preferences. (See "Assigning Preferences" on page 11.)

Protocol	Metric Represents	Range	Unreachable
RIP	Distance (hop-count)	0-15	16
OSPF	Cost of path	1-16777216	Delete
IS-IS	Cost of path	0-254	Delete

This sample table shows the range of possible values for each routing protocol metric and the value used by each protocol to reach a destination.

multi-access networks

Multi-access networks are those physical networks that support the attachment of multiple (more than two) routers. Each pair of routers on such a network is assumed to be able to communicate directly.

multicast

Multicast routing protocols allow packets to be routed to a select set of destinations.

natural mask

A natural mask is a mask of an IP address that is determined by looking at the first two bits of the address. Classful addressing uses only natural masks.

`0x - 255.0.0.0 -` class A

`10 - 255.255.0.0 -` class B

`11 - 255.255.255.0 -` class C

See also RFC 950.

natural network

A natural network is any network that has the same actual and natural masks.

neighbor

A neighbor for one router is another router with which implicit or explicit communication is established by a routing protocol. Neighbors are usually on a shared network, but not always. This term is used mostly in OSPF. The term “neighbor” is usually synonymous with “peer”. (See “peer” on page 176.)

neighboring routers

Neighboring routers are two routers that have interfaces to a common network. On multi-access networks, routers are dynamically discovered by OSPF’s hello protocol.

network

Network refers to any packet-switched network. A network may be specified by its IP address or network name. The host bits in a network specification must be zero. *`Default`* may be used to specify the default network (0.0.0.0).

`network`

Network is either a fully qualified IP host address or an IP prefix. An IP host address should be specified in dotted-quad format, for example, 192.2.0.1. An IP prefix should be specified using only as many numbers as necessary. Both may also be specified as a hexadecimal string preceded by "0x" with an even number of hexadecimal digits of length between two and eight, for example, 0xc0020001. Also allowed is the symbolic value, "**`default`**", which has the value of 0.0.0.0. If "**`options noresolve`**" is not specified, a symbolic network name is used, for example, test.example.net. The numeric forms are much preferred over the symbolic form.

OSPF - Open Shortest Path First

OSPF is one of a class of interior gateway protocols. Open Shortest Path First (OSPF) is a link-state protocol. OSPF is described in more detail in “Chapter 12 Open Shortest Path First (OSPF)” on page 45.

peer

A peer for a router is another router with which implicit or explicit communication is established by a routing protocol. Peers are usually on a shared network, but not always.

This term is used mostly by BGP. Peer is usually synonymous with neighbor. See also “neighbor” on page 176.

port

A *port* is a UDP or TCP port number. Valid values are from 1 through 65535, inclusive.

preference

preference is a value between 0 (zero) and 255 and is used to select between many routes to the same destination. The route with the best (numerically lowest) preference is the active route. The active route is the one installed in the kernel forwarding table and exported to other protocols. Preference zero is usually reserved for routes to directly attached interfaces. A default preference is assigned to each source from which GateD receives routes. (See “Assigning Preferences” on page 11.)

prefix

A prefix is a contiguous mask covering the most significant bits of an address. The prefix length specifies how many bits are covered.

QoS - Quality Of Service

QoS is the level of service provided in terms of delay, throughput, reliability, and cost. QoS is the OSI equivalent of TOS. See also “TOS - Type Of Service” on page 178.

RIP - Routing Information Protocol

RIP is one of a class of interior gateway protocols. RIP assumes that the best route is the one that uses the fewest gateways, i.e., the shortest path, not taking into account congestion or delay on route. (See “Chapter 11 Routing Information Protocol (RIP)” on page 39 for more information about RIP.)

reject route

A reject route is a route with the characteristic that all packets sent along it are discarded. For each such discarded packet, an ICMP network unreachable message is sent to the packet originator.

route filter

A route filter is a description of the characteristics of a set of network addresses. Route filters are used to group routes that require the same policy.

router id

A router id is a 32-bit number assigned to each router running the BGP or OSPF protocol. This number uniquely identifies the router within the autonomous system.

router_id

A *router_id* is an IP address used as a unique identifier assigned to represent a specific router. It is usually the address of an attached interface.

RIB - Routing Information Base

The RIB is the repository of all of GateD’s retained routing information, used to make decisions and as a source for routing information that is propagated.

simplex

A simplex interface is an interface on a broadcast medium that is not capable of receiving packets that it broadcasts. An interface may be marked as simplex either by the kernel or by interface configuration. GateD takes advantage of interfaces that are capable of receiving their own broadcast packets to monitor whether an interface appears to be functioning properly.

time to live (ttl)

The *time to live* of an IP packet is how many hops it can make. Valid values are from 1 through 255 inclusive.

TOS - Type Of Service

The TOS is for Internet service quality selection. The type of service is specified along the abstract parameters of precedence, delay, throughput, reliability, and cost. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses. The vast majority of IP traffic today uses the default type of service. (See also “QoS - Quality Of Service” on page 177.)

unicast

Unicast routing protocols allow packets to be routed to one destination (rather than to several or all possible destinations).

Chapter 38
References

38.1 Requests for Comments (RFCs) by Number

The following is an index of selected RFCs that are of interest to the GateD community, listed in chronological order.

RFC 792 -- http://ietf.org/rfc/rfc792.txt

J. Postel, Internet Control Message Protocol: *DARPA Internet Program Protocol Specification* (September 1981)

RFC 827 -- http://ietf.org/rfc/rfc827.txt

E. Rosen, *Exterior Gateway Protocol (EGP)* (October 1982)

RFC 891 -- http://ietf.org/rfc/rfc891.txt

D. Mills, *DCN Local-network Protocols* (December 1983)

RFC 904 -- http://ietf.org/rfc/rfc904.txt

D. Mills, *Exterior Gateway Protocol Formal Specification* (April 1984)

RFC 1058 -- http://ietf.org/rfc/rfc1058.txt

C. Hedrick, *Routing Information Protocol* (June 1988)

RFC 1105 -- http://ietf.org/rfc/rfc1105.txt

K. Lougheed, Y. Rekhter, *Border Gateway Protocol BGP* (June 1989)

RFC 1112 -- http://ietf.org/rfc/rfc1112.txt

S. Deering, *Host Extensions for IP Multicasting* (August 1989) (Obsoletes RFC1054) (Obsoletes RFC 988)

RFC 1163 -- http://ietf.org/rfc/rfc1163.txt

K. Lougheed, Y. Rekhter, *A Border Gateway Protocol* (BGP) (June 1990)

RFC 1164 -- http://ietf.org/rfc/rfc1164.txt

J. Honig, D. Katz, M. Mathis, Y. Rekhter, J. Yu, *Application of the Border Gateway Protocol in the Internet* (June 1990)

RFC 1195 -- http://ietf.org/rfc/rfc1195.txt

R. Callon, *Use of OSI IS-IS for Routing in TCP/IP and Dual Environments* (December 1990)

RFC 1227 -- http://ietf.org/rfc/rfc1227.txt

M. Rose, *SNMP MUX Protocol and MIB* (May 1991)

RFC 1245 -- http://ietf.org/rfc/rfc1245.txt

J. Moy, *OSPF Protocol Analysis* (July 1991)

RFC 1246 -- http://ietf.org/rfc/rfc1246.txt

J. Moy, *Experience with the OSPF Protocol* (July 1991)

RFC 1247 -- http://ietf.org/rfc/rfc1247.txt

J. Moy, *OSPF Version 2*, (July 1991) (Obsoletes RFC 1131)

RFC 1253 -- http://ietf.org/rfc/rfc1253.txt

F. Baker, R. Coltun, *OSPF Version 2 Management Information Base* (August 1991)

RFC 1256 -- http://ietf.org/rfc/rfc1256.txt

S. Deering, *ICMP Router Discovery Messages* (September 1991)

RFC 1265 -- http://ietf.org/rfc/rfc1265.txt

Y. Rekhter, *BGP Protocol Analysis* (October 1991)

RFC 1266 -- http://ietf.org/rfc/rfc1266.txt

Y. Rekhter, *Experience with the BGP Protocol* (October 1991)

RFC 1267 -- http://ietf.org/rfc/rfc1267.txt

K. Lougheed, Y. Rekhter, *A Border Gateway Protocol 3 (BGP-3)* (October 1991)

RFC 1268 -- http://ietf.org/rfc/rfc1268.txt

P. Gross, Y. Rekhter, *Application of the Border Gateway Protocol in the Internet* (October 1991)

RFC 1269 -- http://ietf.org/rfc/rfc1269.txt

J. Burruss, S. Willis, *Definitions of Managed Objects for the Border Gateway Protocol (Version 3)* (October 1991)

RFC 1321 -- http://ietf.org/rfc/rfc1321.txt

R. Rivest, *The MD5 Message-Digest Algorithm* (April 1992)

RFC 1370 -- http://ietf.org/rfc/rfc1370.txt

Internet Architecture Board, *Applicability Statement for OSPF* (October 1992)

RFC 1388 -- http://ietf.org/rfc/rfc1388.txt

G. Malkin, *RIP Version 2 Carrying Additional Information* (January 1993)

RFC 1389 -- http://ietf.org/rfc/rfc1389.txt

G. Malkin, F. Baker, *RIP Version 2 MIB Extension* (January 1993)

RFC 1397 -- http://ietf.org/rfc/rfc1397.txt

D. Haskin, *Default Route Advertisement In BGP2 And BGP3 Versions Of The Border Gateway Protocol* (January 1993)

RFC 1403 -- http://ietf.org/rfc/rfc1403.txt

K. Varadhan, *BGP OSPF Interaction* (January 1993)

RFC 1448 -- http://ietf.org/rfc/rfc1448.txt

J. Case, K. McCloghrie, M. Rose, S. Waldbusser, *Protocol Operations for Version 2 of Simple Network Management Protocol (SNMPv2)* (April 1993)

RFC 1519 -- http://ietf.org/rfc/rfc1519.txt

V. Fuller, T. Li, K. Varadhan, *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy* (September 1993) (Obsoletes RFC 1338)

RFC 1583 -- http://ietf.org/rfc/rfc1583.txt

J. Moy, *OSPF Version 2* (March 1994)

RFC 1584 -- http://ietf.org/rfc/rfc1584.txt

J. Moy, *Multicast Extensions to OSPF* (March 1994)

RFC 1585 -- http://ietf.org/rfc/rfc1585.txt

J. Moy, *MOSPF: Analysis and Experience* (March 1994)

RFC 1586 -- http://ietf.org/rfc/rfc1586.txt

O. deSouza & M. Rodriques, *Guidelines for Running OSPF Over Frame Relay Networks* (March 1994)

RFC 1587 -- http://ietf.org/rfc/rfc1587.txt

R. Coltun, V. Fuller, *The OSPF NSSA Option* (March 1994)

RFC 1656 -- http://ietf.org/rfc/rfc1656.txt

P. Traina, *BGP-4 Protocol Document Roadmap and Implementation Experience* (July 1994)

RFC 1657 -- http://ietf.org/rfc/rfc1657.txt

S. Willis, J. Burruss, *Definitions of Managed Objects for the Border Gateway Protocol (BGP-4)* (July 1994)

RFC 1700 -- http://ietf.org/rfc/rfc1700.txt

J. Reynolds, J. Postel, *Assigned Numbers* (October 1994) (Obsoletes RFCs 1340, 1060, 1010, 990, 960,943, 923, 900, 870, 820, 790, 776, 770,762, 758,755,750, 739, 604, 503, 433, 349)

RFC 1723 -- http://ietf.org/rfc/rfc1723.txt

G. Malkin, *RIP Version 2 Carrying Additional Information* (November 1994)

RFC 1724 -- http://ietf.org/rfc/rfc1724.txt

G. Malkin, F. Baker, *RIP Version 2 MIB Extension* (November 1994) (Obsoletes RFC 1389)

RFC 1745 -- http://ietf.org/rfc/rfc1745.txt

K. Varadhan, S. Hares, Y. Rekhter, *BGP4/IDRP for IP/OSPF Interaction* (December 1994)

RFC 1765 -- http://ietf.org/rfc/rfc1765.txt

J. Moy, *OSPF Database Overflow* (March 1995)

RFC 1771 -- http://ietf.org/rfc/rfc1771.txt

Y. Rekhter, T. Li, *A Border Gateway Protocol 4 (BGP-4)* (March 1995)

RFC 1772 -- http://ietf.org/rfc/rfc1772.txt

Y. Rekhter, P. Gross, *Application of a Border Gateway Protocol in the Internet* (March 1995)

RFC 1773 -- http://ietf.org/rfc/rfc1773.txt

P. Traina, *Experience with the BGP-4 Protocol* (March 1995)

RFC 1774 -- http://ietf.org/rfc/rfc1774.txt

P. Traina, *BGP-4 Protocol Analysis* (March 1995)

RFC 1793 -- http://ietf.org/rfc/rfc1793.txt

J. Moy, *Extending OSPF to Support Demand Circuits*, (April 1995)

RFC 1850 -- http://ietf.org/rfc/rfc1850.txt

F. Baker, *OSPF Version 2 Management Information Base* (November 1995)

RFC 1863 -- http://ietf.org/rfc/rfc1863.txt

D. Haskin, *BGP/IDRP Route Server Alternative to a Full Mesh Routing* (October 1995)

RFC 1881 -- http://ietf.org/rfc/rfc1881.txt

IAB, IESG, *IPv6 Address Allocation Management* (December 1995)

RFC 1908 -- http://ietf.org/rfc/rfc1908.txt

SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose & S. Waldbusser, *Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework* (January 1996)

RFC 1923 -- http://ietf.org/rfc/rfc1923.txt

J. Halpern, S. Bradner, *RIPv1 Applicability Statement for Historic Status* (March 1996)

RFC 1930 -- http://ietf.org/rfc/rfc1930.txt

J. Hawkinson, *Guidelines for Creation, Selection, and Registration of an Autonomous System (AS)* (March 1996)

RFC 1933 -- http://ietf.org/rfc/rfc1933.txt

R. Gilligan & E. Nordmark, *Transition Mechanisms for IPv6 Hosts and Routers* (April 1996)

RFC 1949 -- http://ietf.org/rfc/rfc1949.txt

S. Jackowski, *Native ATM Support for ST2+* (May 1996)

RFC 1965 -- http://ietf.org/rfc/rfc1965.txt

P. Traina, *Autonomous System Confederations for BGP* (June 1996)

RFC 1966 -- http://ietf.org/rfc/rfc1966.txt

T. Bates, R. Chandra, *BGP Route Reflection: An Alternative to Full Mesh IBGP* (June 1996)

RFC 1997 -- http://ietf.org/rfc/rfc1997.txt

R. Chandra, P. Traina, *BGP Communities Attribute* (August 1996)

RFC 1998 -- http://ietf.org/rfc/rfc1998.txt

E. Chen, T. Bates, *An Application of the BGP Community Attribute in Multi-home Routing* (August 1996)

RFC 2003-- http://ietf.org/rfc/rfc2003.txt

C. Perkins, *IP Encapsulation within IP* (October 1996)

RFC 2024 -- http://ietf.org/rfc/rfc2024.txt

D. Chen, P. Gayek, S. Nix, *Definitions of Managed Objects for Data Link Switching using SMIv2* (October 1996)

RFC 2080 -- http://ietf.org/rfc/rfc2080.txt

G. Malkin, R. Minnear, *RIPng for IPv6* (January 1997)

RFC 2081 -- http://ietf.org/rfc/rfc2081.txt

G. Malkin, *RIPng Protocol Applicability Statement* (January 1997)

RFC 2091 -- http://ietf.org/rfc/rfc2091.txt

G. Meyer, S. Sherry, *Triggered Extensions to RIP to Support Demand Circuits* (January 1997)

RFC2092 -- http://ietf.org/rfc/rfc2092.txt

S. Sherry, G. Meyer, *Protocol Analysis for Triggered RIP* (January 1997)

RFC 2096 -- http://ietf.org/rfc/rfc2096.txt

F. Baker, *IP Forwarding Table MIB* (January 1997)

RFC 2117 -- http://ietf.org/rfc/rfc2117.txt

D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, *Protocol Independent Multicast-Sparse Mode (PIM-SM):Protocol Specification* (June 1997)

RFC 2178 -- http://ietf.org/rfc/rfc2178.txt

J. Moy, *OSPF Version 2* (July 1997) (Obsoletes RFC 1583)

RFC 2185 -- http://ietf.org/rfc/rfc2185.txt

R. Callon, D. Haskin, *Routing Aspects Of IPv6 Transition* (September 1997)

RFC 2189 -- http://ietf.org/rfc/rfc2189.txt

A. Ballardie, S. Reeve & N. Jain, *Core Based Trees (CBT version 2) Multicast Routing -- Protocol Specification* (September 1997)

RFC 2201 -- http://ietf.org/rfc/rfc2201.txt

A. Ballardie, *Core Based Tree (CBT) Multicast Architecture* (July 1996)

RFC 2236 -- http://ietf.org/rfc/rfc2236.txt

W. Fenner, *Internet Group Management Protocol, Version 2* (November 1997)

RFC 2257 -- http://ietf.org/rfc/rfc2257.txt

M. Daniele, B. Wijnen, D. Francisco, *Agent Extensibility (AgentX) Protocol Version 1* (January 1998)

RFC 2270 -- http://ietf.org/rfc/rfc2270.txt

J. Stewart, T. Bates, R. Chandra, E. Chen, *Using a Dedicated AS for Sites Homed to a Single Provider* (January 1998)

RFC 2283 -- http://ietf.org/rfc/rfc2283.txt

T. Bates, R. Chandra, D. Katz, Y. Rekhter, *Multiprotocol Extensions for BGP-4* (February 1998)

RFC 2328 -- http://ietf.org/rfc/rfc2328.txt

J. Moy, *OSPF Version 2* (April 1998)

RFC 2329 -- http://ietf.org/rfc/rfc2329.txt

J. Moy, *OSPF Standardization Report* (April 1998)

RFC 2362 -- http://ietf.org/rfc/rfc2362.txt

D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, L. Wei, *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification* (June 1998)

RFC 2370 -- http://ietf.org/rfc/rfc2370.txt

R. Coltun, *The OSPF Opaque LSA Option* (July 1998)

RFC 2386 -- http://ietf.org/rfc/rfc2386.txt

E. Crawley, R. Nair, B. Rajagopalan, H. Sandick, *A Framework for QoS-based Routing in the Internet* (August 1998)

RFC 2430 -- http://ietf.org/rfc/rfc2430.txt

T. Li, Y. Rekhter, *A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE)* (October 1998)

RFC 2439 -- http://ietf.org/rfc/rfc2439.txt

C. Villamizar, R. Chandra, R. Govindan, *BGP Route Flap Damping* (November 1998)

RFC 2545 -- http://ietf.org/rfc/rfc2545.txt

P. Marques, F. Dupont, *Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing* (March 1999)

RFC 2546 -- http://ietf.org/rfc/rfc2546.txt

A. Durand, B. Buclin, *Bone Routing Practice* (March 1999)

RFC 2547 -- http://ietf.org/rfc/rfc2547.txt

E. Rosen, Y. Rekhter, *BGP/MPLS VPNs* (March 1999)

RFC 2570 -- http://ietf.org/rfc/rfc2570.txt

J. Case, R. Mundy, D. Partain, B. Stewart, *Introduction to Version 3 of the Internet-standard Network Management Framework* (April 1999)

RFC 2571 -- http://ietf.org/rfc/rfc2571.txt

D. Harrington, R. Presuhn, B. Wijnen, *An Architecture for Describing SNMP Management Frameworks* (April 1999)

RFC 2572 -- http://ietf.org/rfc/rfc2572.txt

J. Case, D. Harrington, R. Presuhn, B. Wijnen, *Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)* (April 1999)

RFC 2573 -- http://ietf.org/rfc/rfc2573.txt

D. Levi, P. Meyer, B. Stewart, *SNMP Applications* (April 1999)

RFC 2574 -- http://ietf.org/rfc/rfc2574.txt

U. Blumenthal, B. Wijnen, *User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)* (April 1999)

RFC 2575 -- http://ietf.org/rfc/rfc2575.txt

B. Wijnen, R. Presuhn, K. McCloghrie, *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)* (April 1999)

RFC 2588 -- http://ietf.org/rfc/rfc2588.txt

R. Finlayson, *IP Multicast and Firewalls* (May 1999)

RFC 2622-- http://ietf.org/rfc/rfc2622.txt

C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, M. Terpstra, *Routing Policy Specification Language (RPSL)* (June 1999)

RFC 2650 -- http://ietf.org/rfc/rfc2650.txt

D. Meyer, J. Schmitz, C. Orange, M. Prior, C. Alaettinoglu, *Using RPSL in Practice* (August 1999)

RFC 2676 -- http://ietf.org/rfc/rfc2676.txt

G. Apostolopoulos, D. Williams, S. Kamat, R. Guerin, A. Orda, T. Przygienda, *QoS Routing Mechanisms and OSPF Extensions* (August 1999)

RFC 2702 -- http://ietf.org/rfc/rfc2702.txt

D. Awduche, J. Malcolm, J. Agogbua M. O'Dell J. McManus, *Requirements for Traffic Engineering Over MPLS* (September 1999)

RFC 2710 -- http://ietf.org/rfc/rfc2710.txt

S. Deering, W. Fenner, B. Haberman, *Multicast Listener Discovery (MLD) for IPv6* (October 1999)

RFC 2715 -- http://ietf.org/rfc/rfc2715.txt

D. Thaler, *Interoperability Rules for Multicast Routing Protocols* (October 1999)

RFC 2725 -- http://ietf.org/rfc/rfc2725.txt

C. Villamizar, C. Alaettinoglu, D. Meyer, S. Murphy, *Routing Policy System Security* (December 1999)

RFC 2740 -- http://ietf.org/rfc/rfc2740.txt

R. Coltun, D. Ferguson, J. Moy, *OSPF for IPv6* (December 1999)

RFC 2741 -- http://ietf.org/rfc/rfc2741.txt

M. Daniele, B. Wijnen, M. Ellison, D. Francisco, *Agent Extensibility (AgentX) Protocol Version 1* (January 2000)

RFC 2742 -- http://ietf.org/rfc/rfc2742.txt

L. Heintz, S. Gudur, M. Ellison, *Definitions of Managed Objects for Extensible SNMP Agents* (January 2000)

RFC 2763 -- http://ietf.org/rfc/rfc2763.txt

N. Shen, H. Smit., *Dynamic Hostname Exchange Mechanism for IS-Iietf.orgS* (February 2000)

RFC 2796 -- http://ietf.org/rfc/rfc2796.txt

T. Bates, R. Chandra, E. Chen, BGP *Route Reflection - An Alternative to Full Mesh IBGP* (April 2000)

RFC 2842 -- http://ietf.org/rfc/rfc2842.txt

R. Chandra, J. Scudder, *Capabilities Advertisement with BGP-4* (May 2000)

RFC 2858 -- http://ietf.org/rfc/rfc2858.txt

T. Bates, Y. Rekhter, R. Chandra, D. Katz, *Multiprotocol Extensions for BGP-4* (June 2000) (Obsoletes RFC 2283)

RFC 2933 -- http://ietf.org/rfc/rfc2933.txt

K. McCloghrie, D. Farinacci, D. Thaler, *Internet Group Management Protocol MIB* (October 2000)

RFC 2934 -- http://ietf.org/rfc/rfc2934.txt

K. McCloghrie, D. Farinacci, D. Thaler, B. Fenner, *Protocol Independent Multicast MIB for IPv4* (October 2000)

RFC 2966 -- http://ietf.org/rfc/rfc2966.txt

T. Li, T. Przygienda, H. Smit, *Domain-wide Prefix Distribution with Two-Level IS-IS* (October 2000)

RFC 2973 -- http://ietf.org/rfc/rfc2973.txt

R. Balay, D. Katz, J. Parker, *IS-IS Mesh Groups* (October 2000)

RFC 3065 -- http://ietf.org/rfc/rfc3065.txt

P. Traina, D. McPherson, J. Scudder, *Autonomous System Confederations for BGP* (February 2001) (Obsoletes RFC 1965)