

Command Reference

GateD V.9.3.2



Copyright © NextHop Technologies, 2002

Copyright Notice

Except as stated herein, none of the material provided as a part of this document may be copied, reproduced, distributed, republished, downloaded, displayed, posted or transmitted in any form or by any means, including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of NextHop Technologies, Inc. Permission is granted to display, copy, distribute and download the materials in this document for personal, non-commercial use only, provided you do not modify the materials and that you retain all copyright and other proprietary notices contained in the materials unless otherwise stated. No material contained in this document may be "mirrored" on any server without written permission of NextHop. Any unauthorized use of any material contained in this document may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes. Permission terminates automatically if any of these terms or conditions are breached. Upon termination, any downloaded and printed materials must be immediately destroyed.

Trademark Notice

The trademarks, service marks, and logos (the "Trademarks") used and displayed in this document are registered and unregistered Trademarks of NextHop in the US and/or other countries. The names of actual companies and products mentioned herein may be Trademarks of their respective owners. Nothing in this document should be construed as granting, by implication, estoppel, or otherwise, any license or right to use any Trademark displayed in the document. The owners aggressively enforce their intellectual property rights to the fullest extent of the law. The Trademarks may not be used in any way, including in advertising or publicity pertaining to distribution of, or access to, materials in this document, including use, without prior, written permission. Use of Trademarks as a "hot" link to any website is prohibited unless establishment of such a link is approved in advance in writing. Any questions concerning the use of these Trademarks should be referred to NextHop at U.S. +1 734 222 1600.

Contents

Chapter 1	
About this Manual	1
Overview	1
Audience	1
GateD Command Reference Sections	1
Chapter 2	
Trace Statements and Global Options	3
traceoptions	3
Chapter 3	
Directive Statements	7
%directory	7
%include	8
Chapter 4	
Options Statements	9
mark	9
noresolv	10
nosend	11
options	12
syslog	13
Chapter 5	
Interface Statements	15
aliases-nexthop	15
alias primary	18
AS	19
blackhole	20
broadcast	21
define	22
disable	25
down preference	26
enable	27
interface	28
interfaces	30
multicast nomulticast	34
netmask	35
options	36
passive	38
preference	39

reject	40
remote	41
scaninterval	42
simplex	44
strictinterfaces	45
tunnel	46
unicast nunicast	47
Chapter 7	
Routing Information Protocol (RIP)	49
authentication	49
broadcast, nobroadcast	53
defaultmetric	55
expire-time	56
ignorehostroutes	57
interface	58
metricin	60
metricout	62
nocheckzero	64
preference	65
query authentication	67
rip	69
ripin, noripin	70
ripout, noripout	72
secondary authentication	74
sourcegateways	76
traceoptions	78
trustedgateways	80
update-time	82
version	83
Chapter 6	
Definitions	87
autonomoussystem	87
confed-id	89
martians	90
routerid	93
Chapter 8	
Open Shortest Path First Protocol (OSPF)	95
advertise-subnet	95
always-update-summary	97
area	98
auth	99
backbone	101
cost	102
defaults	103
disable	105
enable	106
hellointerval	107
inherit-metric	109

interface	110
networks	114
nssa	116
nssa-cost	117
nssa-inherit-metric	118
nssa-preference	119
nssa-type	120
nssanetworks	121
opaque-capability	123
ospf	124
passive	125
pollinterval	126
preference	127
priority	128
retransmitinterval	130
rfc1583compatibility	131
ribs	132
routerdeadinterval	133
routers	135
strict-routers	137
stub	138
stubhosts	139
stubnetworks	141
summaryfilters	143
tag	145
traceoptions	146
transitdelay	148
type	150
virtuallink	151
Chapter 9	
Intermediate System to Intermediate System (IS-IS)	153
area	153
area auth	155
auth	157
config-time	160
csn-interval	161
disable	163
dis-hello-interval	164
domain auth	166
domain-wide	168
enable	169
es-config-time	170
export-defaults	171
extended-metrics	174
external preference	175
hello-interval	176
hello-multiplier	178
hold-time	180
hostname	181

inet	182
inet6	183
interface	184
isis	185
level	187
lsp-interval	189
max-burst	190
mesh-blocked	192
mesh-group	193
metric	194
overload-bit	196
passive	197
periodic-csn	198
preference	199
priority	200
psn-interval	202
require-snp-auth	203
retransmit-interval	204
rfc1195-metrics	205
ribs	206
spf-interval	207
start-accept	208
start-generate	210
stop-accept	212
stop-generate	214
summary-filter	216
summary-originate	219
systemid	222
traceoptions	224
Chapter 10	
Border Gateway Protocol (BGP)	227
allow	227
ascount	230
bgp	232
clusterid	234
comm	236
confed	238
defaultmetric	241
discard-nonprefixed-confederations	242
gateway	244
group type	247
holdtime	250
ignorefirstashop	252
ignore-nonprefixed-confederations	254
keep	256
keepalivesalways	258
localas	260
localtcp	261
logupdown	263

med	265
metricout	267
nexthopself	269
noaggregatorid	271
nogendefault	273
no-mp-nexthop	274
nov4asloop	276
open-on-accept	277
outdelay	278
passive	279
peer	281
preference	283
preference2	285
recvbuffer	287
reflector-client [no-client-reflect]	288
routetopeer	290
sendbuffer	291
setpref	292
showwarnings	294
traceoptions	295
ttl	297
Chapter 11	
Router Discovery	299
address	299
interface (client)	301
interface (server)	303
preference	305
routerdiscovery client	306
routerdiscovery server	308
traceoptions	311
Chapter 12	
Internet Control Message Protocol (ICMP) Statement	313
icmp	313
traceoptions	315
Chapter 13	
Redirect Processing	319
interface	319
preference	321
redirect	322
traceoptions	324
trustedgateways	325
Chapter 14	
Kernel Interface	327
background	327
flash	329
kernel	331
options	334

remnantholdtime	336
routes	337
traceoptions	339
Chapter 15	
Static Routes	343
as	343
blackhole	344
default	345
gateway	346
host	348
interface	349
multicast	351
noinstall	352
preference	353
reject	355
retain	356
static	357
unicast	359
Chapter 16	
Distance Vector Multicast Routing Protocol (DVMRP)	361
defaultmetric	361
disable	363
dvmrp	364
enable	366
interface	367
metric	368
nodvmrpout	369
noretransmit	370
preference	371
prune-lifetime	372
routing-only	373
traceoptions	374
tunnel-compatible	376
Chapter 17	
Protocol Independent Multicast (PIM)	379
assert-holdtime	379
boundary	381
bsr	382
bsr-holdtime	385
bsr-period	386
crp	388
crp-adv-period	391
crp-holdtime	392
dr-switch-immediate	393
group	395
hello-holdtime	397
hello-interval	399
hello-priority	401

interface	403
jp-holdtime	405
jp-interval	407
mrt-period	409
mrt-spt-mult	410
mrt-stale-mult	412
pim	414
priority	416
probe-period	419
reg-sup-timeout	420
rp-switch-immediate	422
sparse	424
static-rp	425
threshold	426
threshold-dr	428
threshold-rp	430
traceoptions	432
wholepkt-checksum	434
Chapter 18	
Multi-Protocol - Border Gateway Protocol (MPBGP)	435
allow	435
ascount	438
caps	440
clusterid	442
comm	444
confed	446
defaultmetric	449
discard-nonprefixed-confederations	450
gateway	452
group type	455
holdtime	458
ignorefirstashop	460
ignore-nonprefixed-confederations	462
interface	464
keep	465
keepalivesalways	467
localas	469
localtcp	470
localv4addr	472
localv6addr	474
logupdown	476
med	478
metricout	480
mpbgp	482
nexthopself	483
noagggregatorid	485
nogendefault	486
nov4asloop	487
open-on-accept	488

outdelay	489
passive	490
peer	491
preference	493
preference2	495
recvbuffer	497
reflector-client [no-client-reflect]	498
remotev4addr	500
remotev6addr	502
routetopeer	504
sendbuffer	505
setpref	506
showwarnings	507
traceoptions	508
ttl	510
Chapter 19	
Multicast Source Discovery Protocol (MSDP)	511
connect-retry-period	511
default-rpf-peer	513
keepalive-period	514
msdp	515
msdp-draft-6-compatible	516
peer	517
peer-holdtime	519
pim-filter	520
sa-cache-timeout	521
sa-filter	522
sa-holddown	524
static-rpf-peer	525
traceoptions	526
Chapter 20	
Internet Group Management Protocol (IGMP)	527
enable disable	527
igmp	529
interface	531
last-mem-query-intvl	533
max-response-time	536
nosend	538
query-interval	539
robustness	541
traceoptions	543
Chapter 21	
Multicast Statement	545
boundary	545
interface	547
multicast	548

Chapter 22	
Mstatic Statement	549
disable	549
enable	551
interface	552
join	553
mstatic	554
Chapter 23	
Routing Information Protocol, next generation (RIPng)	557
defaultmetric	557
expire-time	559
interface	560
metricin	562
metricout	564
preference	566
ripng	568
ripin, noripin	570
ripout, noripout	571
traceoptions	573
update-time	575
Chapter 24	
Route Filtering	577
all	577
between	579
default	580
exact	581
host	582
network	583
refines	585
route filter	586
Chapter 25	
Matching AS Paths	589
aspath	589
origin	591
Chapter 26	
BGP Communities	593
comm	593
comm-add	595
comm-delete	596
community_list	597
ext-comm	599
ext-comm-add	600
ext-comm-delete	601
extended_community_list	602
Chapter 27	
Route Importation	605

fromribs	605
import proto bgp	607
import proto ospfase	610
import proto redirect	612
import proto rip	614
import proto ripng	616
preference	618
restrict	619
tag	620
toribs	621
Chapter 28	
Route Exportation	623
export proto bgp	623
export proto isis	627
export proto ospfase	630
export proto ospfnssa	633
export proto rip	635
export proto ripng	638
fromribs	641
proto aggregate	643
proto bgp	646
proto direct	649
proto isis	652
proto kernel	655
proto ospf	657
proto ospfase	660
proto rip	663
proto ripng	666
proto static	669
Chapter 29	
Route Aggregation and Generation	673
aggregate	673
as	675
aspath	676
brief	677
generate	678
preference	680
proto aggregate	682
proto all	684
proto bgp	686
proto direct	688
proto isis	690
proto kernel	692
proto ospf	694
proto ospfase	696
proto rip	698
proto ripng	700
proto static	702

restrict	704
toribs unicast multicast	705
Chapter 30	
Route Flap Damping	707
dampen-flap	707
keep-history	709
max-flap	710
reach-decay	711
reuse-below	712
suppress-above	713
unreach-decay	714
Chapter 31	
SNMP Multiplexing (SMUX)	715
password	715
port	716
smux	717
traceoptions	718

Chapter 1

About this Manual

1.1 Overview

This manual lists GateD commands alphabetically within protocol sections. For example, if you are looking for the **authentication** command in RIP, look in Chapter 7, RIP, under the A's.

For information about how to install GateD, see *Installing GateD*. For information on protocol syntax or a general overview of a protocol, see *Configuring GateD*. For information on the operation of GateD, including options, signals, and various support programs, see *Operating GateD*. (Utilities covered include gdc, rip query, gated, gii, and ospfmon.)

1.2 Audience

This manual is written for operators who are configuring GateD to route packets. It is designed as a companion to the *Configuring GateD* manual, and explains each GateD command in greater detail. You will need to understand basic routing concepts and UNIX commands to understand this manual.

1.3 GateD Command Reference Sections

In general, command entries of this manual have eight sections:

- Name
- Syntax
- Parameters
- Description
- Default
- Context
- Examples
- See Also

1.3.1 Name

The Name section lists the name and a short description of the command. For example, the **key** command in RIP:

key - sets a RIP MD5 key

1.3.2 Syntax

The Syntax section lists all valid syntax formats. For example, the **key** command in RIP has two valid syntax formats:

```
key password id med5idnum ;
```

```
key password id med5idnum { start-generate date-time ; stop-generate  
    date-time; start-accept date-time; stop-accept date-time; } ;
```

key is a command and is shown in **bold** font; GateD expects the term as shown. *password* is a variable and is shown in *italic* font; GateD expects you to supply a password.

1.3.3 Parameters

The Parameters section lists each possible parameter for the command, a description of what sort of parameter GateD expects (for example, decimal number or hexadecimal number), and the range of values GateD expects. (For example, 0.0.0.0 to 255.255.255.255 is the range for a RIP *password*.)

1.3.4 Description

The Description section includes a detailed description of the command.

1.3.5 Default

The Default section includes the default value(s) of the command and its parameters.

1.3.6 Context

The Context section lists the valid locations in which GateD expects this command to be used. For example, the RIP **key** command is expected on the **rip authentication** statement or the **rip query** statement.

1.3.7 Examples

The Examples section lists valid syntax including this command.

1.3.8 See Also

The See Also section lists other commands or sections of this guide or *Configuring GateD* that might be useful. In addition, other publicly available documents, such as RFCs, may be listed here.

Chapter 2

Trace Statements and Global Options

traceoptions

Name

traceoptions - specifies packet tracing options

Syntax

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

Parameters

trace_file - specifies the file to receive tracing information. If this file name does not begin with a slash ('/'), the directory where GateD was started is prepended to the name. Specifying "**stderr**" writes trace output to standard error.

replace - specifies to start tracing by truncating an existing file. The default is to append to an existing file.

size *tracesize* [**k** | **m**] **files** *tracefiles* - limits the maximum size of the trace file to the specified *tracesize* (minimum 10k). When the trace file reaches the specified size, the file is renamed to file.0, then file.1, then file.2, up to the maximum number of files as specified by *tracefiles*. (The minimum specification is 2.)

k - specifies the file size in kilobytes

m - specifies the file size in megabytes

files *tracefiles* - specifies the maximum number of files

nostamp - specifies that a timestamp should not be prepended to all trace lines

trace_options - described below

The *trace_options* that have only global significance are:

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

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 - 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 - 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.

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

none - specifies that all tracing should be turned off for this protocol or peer

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

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

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

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

state - specifies to trace state machine transitions in the protocols

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

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

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

Tracing of packets is very flexible. For any given protocol, there are one or more options for tracing packets. 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 - must be specified before **send** or **receive**. 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.

send or **receive** - limit the tracing to packets sent or received. If neither is specified, both sent and received packets will be traced.

except trace_options - used to enable a broad class of tracing and then disable more specific options

Description

Trace statements control tracing options. GateD's tracing options can be configured at many levels. Tracing options include the file specifications 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.

Not all of the above *trace_options* apply to all of the protocols. In some cases, their use would not make sense (for instance, RIP does not have a state machine) and, in some cases, the requested tracing has not been implemented. Currently, you cannot specify packet tracing from the command line because a global option for packet tracing could create too much output.

When protocols inherit their tracing options from the global tracing options, tracing options that do not 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 specified in the configuration file 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. When more than one **traceoptions** line is used in a section, the "last" **traceoptions** line to be parsed by GateD is the one that takes effect. In the case of global tracing, any trace files specified in any **traceoptions** line will be created, but tracing will cease for that file when the next **traceoptions** line is parsed.

The **size** and **file** options in subsequent **traceoptions** statements with the same file name modify the previously set values for the files. If a subsequent **traceoptions** statement refers to the same file but does not specify **size** and **file** values, the size limit for the file is set to unlimited. Setting **size** to 0 also sets the size limit to unlimited. In this latter case, the **files** keyword and value can be omitted.

Defaults

- Tracing is written to standard error.
- There is no limit on the file size.
- **nostamp** is disabled.
- There are no trace options.

Context

global

Examples

Example 1

The first example traces everything but policy to "icmp/GateD.log" (iflist is superfluous).

```
traceoptions "/tmp/GateD.log" replace size 100 K files 3
iflist all except policy;
```

Example 2

```
traceoptions "log" size 1024k files 2;
# Once the log file reaches 1 megabyte in length, it is
# moved to log.0, and the log is recreated.
#
```

```
# Again, when the log file file reaches 1 megabyte in length,  
# log.0 is renamed log.1, log is renamed log.0,  
# and the log is recreated.  
#  
# Again, when the log file reaches 1 megabyte in length,  
# log.1 is deleted, log.0 is renamed log.1, log is renamed log.0,  
# and the log is recreated.
```

Example 3

```
traceoptions "/tmp/GateD.log"  
# The log file is /tmp/GateD.log. It will grow in size until  
# space on /tmp is depleted
```

Example 4

```
traceoptions policy;  
traceoptions all except general;
```

See Also

rip on page 49
bgp on page 227
ospf on page 95
isis on page 153
icmp on page 313
redirect on page 319
kernel on page 327
dvmrp on page 361
pim on page 379
mpbgp on page 435
igmp on page 527
smux on page 715

Chapter 3

Directive Statements

%directory

Name

%directory - defines the directory in which the included files are stored

Syntax

%directory "*directory*"

Parameters

directory - a directory containing GateD configuration files

Description

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 included file names.

Only a single directory name can be specified. If more than one **%directory** statement is encountered, the last one encountered is used to find subsequent include files.

Defaults

defaults to the directory from which GateD is run

Context

global

Examples

```
%directory "/usr/local/gatedConfiguration"
```

See Also

"Chapter 5 Directive Statements" on page 19 of *Configuring GateD*

%include on page 8

%include

Name

%include - identifies an include file

Syntax

%include *"filename"*

Parameters

filename - the name of the file being included

Description

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 ten levels is supported. The maximum nesting level can be increased by changing the definition of **FI_MAX** in `parse.h`.

Defaults

none

Context

global

Examples

```
%include "traceOptions"
```

```
%include "optionsStatements"
```

```
%include "protocolStatements"
```

```
%include "staticStatements"
```

```
%include "controlStatements"
```

See Also

"Chapter 5 Directive Statements" on page 19 of *Configuring GateD*

%directory on page 7

Chapter 4

Options Statements

mark

Name

mark - causes GateD to output a message to the trace log at the specified interval

Syntax

mark *time*

Parameters

time - the time interval between "mark messages" in the form of seconds, minutes:seconds, or hours:minutes:seconds

Description

mark causes GateD to output a message to the trace log at the specified interval. **mark** can be used as one method of determining whether GateD is still running.

Defaults

options mark 0; # disable mark messages

Context

options statement

Examples

The following example causes GateD to generate a mark message every 5 minutes:

```
options mark 5:0 ;
```

See Also

options on page 12

noreolv

Name

noreolv - prevents mapping symbolic names to IP addresses

Syntax

noreolv

Parameters

none

Description

By default, GateD will try to resolve symbolic names into IP addresses by using the `gethostbyname()` and `getnetbyname()` library calls. These calls usually use the Domain Name System (DNS) instead of the host's local host and network tables.

If there is insufficient routing information to send DNS queries, GateD will deadlock during startup. **noreolv** can be used to prevent these calls. Symbolic names will result in configuration file errors.

Defaults

This option is disabled. Symbolic host names are mapped to IP addresses.

Context

options statement

Examples

```
options noreolv;
```

See Also

options on page 12

nosend

Name

nosend - prevents GateD from sending packets

Syntax

nosend

Parameters

none

Description

nosend makes it possible to run GateD on a live network to test protocol interactions without actually participating in the routing protocols. The packet traces in the GateD log can be examined to verify that GateD is functioning properly.

nosend applies to OSPF, ISIS, and all other protocols that send UDP packets.

Defaults

This option is disabled. Packets are sent by default, i.e., GateD participates in the routing protocols.

Context

options statement

Examples

```
options nosend;
```

See Also

options on page 12

options

Name

options - allow specification of certain global options

Syntax

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

Parameters

nosend

noresolv

syslog

mark

Description

options statements let you specify certain global options. If used, **options** must appear before any other type of configuration statement in the `gated.conf` file.

Defaults

nosend, **noresolv**, and **mark** are disabled by default.

```
options syslog upto info;
```

Context

global

Examples

```
options nosend noresolv syslog notice mark 5:00;
```

See Also

“Route Aggregation” on page 673

syslog

Name

syslog - sets GateD logging level

Syntax

```
syslog ( [ upto ] log_level ... )
```

Parameters

The values for *log_level* are (high to low):

Table 1: Values for *log_level*

log_level value	setlogmask term	Definition
emerg (high)	LOG_EMERG	A panic condition
alert	LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database
crit	LOG_CRIT	Critical conditions, e.g., hard disk errors
err	LOG_ERR	Errors
warning	LOG_WARNING	Warning messages
notice	LOG_NOTICE	Conditions that are not error conditions but should possibly be handled individually
info	LOG_INFO	Informational messages
debug (low)	LOG_DEBUG	Messages that contain information normally of use only when debugging a program

Description

syslog controls the amount of data GateD logs via syslog on systems where **setlogmask()** is supported. The available logging levels and other terminology are as defined in the **setlogmask(3)** man page.

Defaults

```
options syslog upto info;
```

Context

`options` statement

Examples

The following example logs `emerg`, `alert`, and `crit` only.

```
options syslog upto crit;
```

The above example can also be written the following way.

```
options syslog emerg crit alert ;
```

See Also

`options` on page 12

Chapter 5

Interface Statements

aliases-nexthop

Name

aliases-nexthop - specifies the address that GateD will install as the next hop for the route associated with this interface

Syntax

```
aliases-nexthop ( primary | lowestip ) ;
```

ALSO

```
aliases-nh ( primary | lowestip ) ;
```

Parameters

primary - The primary interface address (default) will be installed as the next hop for the route associated with this interface. This is the recommended setting for all interfaces and is the default.

lowestip - The address with the lowest IP address will be installed as the next hop for the route associated with this interface.

Description

aliases-nexthop specifies which address GateD will install as the next hop for the route associated with an interface. 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)` UNIX 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.

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 ffffffff00 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 in *Configuring GateD* 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.

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.

Defaults

```
aliases-nexthop primary ;
```

Context

```
interfaces interface statement
```

```
interfaces options statement
```

Examples

```
interfaces { options alias-nexthop lowestip ; } ;
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

`interfaces` on page 30

`options` on page 36

alias primary

Name

alias primary - specifies primary addresses for an interface

Syntax

```
alias primary address mask mask ;
```

Parameters

address - the interface's primary network address

mask - a mask specifying a subnet

Description

alias primary is used to override the default selection of the primary logical interfaces on a given physical interface. There is exactly one primary interface for each subnet defined on a given physical interface. By default, GateD selects the first logical interface read from the kernel for each subnet defined on the physical interface as a primary interface. **alias primary** provides a way of specifying which of the logical interfaces that share the same subnet is to be selected as primary for that subnet.

Defaults

A default primary address is selected by GateD for each subnetwork defined on a physical interface. The default primary addresses are the addresses of the first interfaces read from the kernel for each subnet on a physical interface.

Context

interfaces *interface* statement

Examples

```
interfaces {  
    interface ppp1 alias primary 142.77.34.184 mask 255.255.192.0;  
};
```

See Also

aliases-nexthop on page 15

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

AS

Name

as - specifies the autonomous system (AS) used to create an AS path associated with the route created from the definition of this interface

Syntax

```
AS autonomoussystem ;
```

Parameters

autonomoussystem - AS numbers are currently in the range 1 to 65535, inclusive.

Description

as specifies the autonomous system that will be used to create an AS path associated with the route created from the definition of this interface. The autonomous system number of the router running GateD is specified in the global AS statement. The autonomous system numbers of BGP's peers is specified in BGP configuration. An interface defines a route, known as an "interface route". Interface routes are also known as direct routes. Specifying an AS number on the interface clause will cause the direct route corresponding to the interface to be generated with a non-empty AS path. If this route is then exported into BGP, update messages advertising this route will include the specified AS in their AS path. Normally, the interface AS number is not set.

Defaults

By default, no AS number is associated with an interface.

Context

interfaces *interface* statements

Examples

```
interfaces {  
    interface ppp1  
        AS 1439;  
};
```

See Also

bgp statement on page 232

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

blackhole

Name

blackhole - causes the local address of this interface to be used as the next hop for blackhole routes.

Syntax

```
blackhole ;
```

Parameters

none

Description

blackhole specifies that this interface is the blackhole pseudo-interface supported by some older kernels. This interface must be of type "loopback" and should discard any packets sent to it without sending back "unreachable messages". This option is archaic and should no longer be necessary. Modern kernels recognize the RTF_BLACKHOLE flag associated with blackhole routes and directly perform the desired action of silently discarding packets sent to those routes.

Defaults

blackhole is disabled by default.

Context

interfaces interface statement

Examples

```
interfaces {  
    interface bh0  
        blackhole;  
}
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

"Chapter 19 Static Routes" on page 101 of *Configuring GateD*

reject on page 40

broadcast

Name

broadcast - defines the interface as broadcast capable

Syntax

```
broadcast address ;
```

Parameters

address - the broadcast address

Description

broadcast defines the interface as broadcast capable (for example, Ethernet or Token Ring) and specifies the broadcast address. The broadcast address is typically the subnet address followed by all 1s. It is an error to specify a broadcast address on a point-to-point link.

Defaults

If the interface exists, the default broadcast address is that of the interface.

Context

interfaces **define** statement

Examples

```
interfaces {  
    define subnet local 192.168.13.129  
        netmask 255.255.255.252  
        broadcast 192.168.13.131 ;  
}
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

define

Name

`define`

Syntax

```
define (subnet | p2p) local address [ options ] ;
```

Parameters

subnet - defines a subnet type interface

p2p - defines a point-to-point link type interface

address - the local address to be used for the interface

options - options for this interface

Description

The **define** statement is used to alter the existing interface configuration, and to provide a default configuration should the specified interface(s) be UNIX configured, using `ifconfig(1)`. **define** may specify interfaces that might not be present when GateD is started, enabling the configuration file to refer to them when **strictinterfaces** is defined.

subnet implies a broadcast or NBMA interface and defines a subnet (and refers to all interfaces on the subnet). When an interface comes up on this subnet, its configuration will be merged with the options present in the **define** clause.

p2p defines a point-to-point link. The configuration of a point-to-point interface will be merged with the options present in the **define** clause.

Possible **define options** are:

broadcast address - **broadcast** defines the interface as broadcast capable (for example, Ethernet or Token Ring) and specifies the broadcast address. The broadcast address is typically the subnet address followed by all 1s. It is an error to specify a broadcast address on a point-to-point link.

remote address - The remote address specifies the remote address of the logical point-to-point link. It is an error to specify a remote address if **subnet** is specified.

tunnel encapsulation_protocol - **tunnel** defines the encapsulation protocol to use for a point-to-point tunnel. Currently, only IPIP encapsulation (RFC 2003) is recognized. (IPIP encapsulation is used by multicast tunnels supported by kernels.) RFC 2003 is available at: <http://ietf.org/rfc/rfc2003.txt>

netmask address - **netmask** defines the netmask of a logical subnet. It is an error to specify a netmask address on a point-to-point link.

[no]multicast - **[no]multicast** explicitly enables/disables multicast on the interface. By default, IPIP tunnels are multicast, nunicast interfaces.

[no]unicast - **[no]unicast** explicitly enables/disables unicast routing on the interface. By default, IPIP tunnels are multicast, nunicast interfaces.

Defaults

If the interface exists, defaults come from it. An interface not defined as broadcast or point-to-point is assumed to be NBMA, such as an X.25 network. Remaining defaults are currently not part of the API specification.

Context

`interfaces` statement

Examples

Example 1

```
interfaces {
    interface ppp2;
    define p2p local 176.144.13.44 remote 176.144.12.92 nomulticast;
};
```

Example 2

This `define` configures a multicast-only IP-in-IP tunnel usable by routing protocols for the multicast RIB. (See Chapter 1, Section 8 “Multiple RIBs” on page 33 of *Configuring GateD* for more information about multicast RIBs.) Note that the keywords `multicast` `nounicast` here are redundant with the defaults for `tunnel ipip`. In fact, the standard multicast kernel cannot support any other combination.

```
interfaces {
    define p2p local 198.108.60.89 remote
        141.213.10.41 multicast nounicast
        tunnel ipip;
};
```

Example 3

This `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.)

```
interfaces {
    define subnet local 192.168.12.114 netmask
        255.255.255.0;
};
```

Example 4

This `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 as a subnet type interface with a netmask of 255.255.255.252 and a broadcast address of 192.168.13.131.

```
interfaces {  
    define subnet local 192.168.13.129 netmask  
        255.255.255.252  
        broadcast 192.168.13.131;  
};
```

Example 5

This **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 is not actually a point-to-point link. (If it is actually a subnet, this requires that the configured remote point-to-point address fall within the actual subnet prefix.)

```
interfaces {  
    define p2p local 192.168.13.114 remote  
        192.168.13.116;  
};
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

interfaces on page 30

disable

Name

disable - indicates that GateD will disable operation on this interface even if the kernel reports it as up

Syntax

```
disable ;
```

Parameters

none

Description

This option indicates that GateD shall disable operation on this interface even if the kernel reports it as up.

Defaults

Interfaces are enabled by default.

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface all enable ;  
    interface ex0 disable ;  
} ;
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

enable on page 27

interfaces on page 30

down preference

Name

down preference - sets the preference for routes to this interface when GateD does not believe it to be functioning properly, but the kernel does not indicate that it is down

Syntax

```
down preference downpreference ;
```

Parameters

downpreference - Preferences are in the range 0 to 255, inclusive, with 0 being the highest preference a route can have.

Description

Sets the preference for routes to this interface when GateD does not believe it to be functioning properly, but the kernel does not indicate that it is down. See **passive** on page 38 for a description of “not functioning properly”.

Defaults

```
down preference 120;
```

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface all down preference 150;  
}
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

interfaces on page 30

passive on page 38

preference on page 39

enable

Name

enable - indicates that GateD will enable this interface if it is reported as up by the kernel

Syntax

```
enable ;
```

Parameters

none

Description

This option exists for symmetry with the **disable** option. When present, it indicates that this interface shall be enabled by GateD if it is reported as up by the kernel.

Defaults

Interfaces are enabled by default.

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface all disable ;  
    interface ex0 enable ;  
}
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

disable on page 25

interfaces on page 30

interface

Name

interface - sets or modifies interface characteristics for the specified interfaces

Syntax

```
interface interface_list [ options ] ;
```

Parameters:

interface_list - one or more of the following:

all - all available interfaces

wildcard - **interface** *wildcard* refers to all the interfaces of the same type. UNIX interfaces consist of the name of the device driver, such as *ie*, and a unit number, such as 0, 5, or 22. References to the name contain only alphabetic characters and match any interfaces that have the same alphabetic part. For example, *ie* on a Sun would refer to all Interlan Ethernet interfaces, and *le* would refer to all Lance Ethernet interfaces. However, *ie* would not match *iox0*.

name - **interface** *name* refers to a specific interface, usually one physical interface. This name is specified as an alphabetic part followed by a numeric part. **interface** *name* will match one specific interface. Be aware that on many systems, more than one protocol (i.e., IP) address can be on a given physical interface. For example, *ef1* will match an interface named *ef1*, but not an interface named *ef10*.

address - **interface** *address* matches one specific interface. The reference can be by protocol address (for example, 10.0.0.51), or by symbolic hostname (for example, *nic.ddn.mil*). Note that a symbolic hostname reference is valid only when it resolves to only one address. Use of symbolic hostnames is not recommended.

This address specifies the unique address of the interface referred to. The unique address is the local address for all but point-to-point interfaces and is the remote address for point-to-point interfaces.

local address - **local address** is the same as *address* except that the local address (not remote) is matched.

remote address - **remote address** is the same as *address* except that the remote address (not local) is matched.

options - one or more of the following:

preference - specifies the preference of the interface when up

down preference - specifies the preference of the interface when down

enable - specifies that the interface is to be used by GateD

disable - specifies that the interface is not used by GateD

passive - specifies that the interface is not to be marked as down due to inactivity

simplex - indicates that the interface does not receive its own multicast packets

reject - specifies that the interface is to be used by the kernel for reject routes

blackhole - specifies that the interface is to be used by the kernel for blackhole routes

AS *autonomoussystem* - associates the value of *autonomoussystem* with the route created for this interface

alias primary *address mask mask* - specifies the primary address for this interface

aliases-nexthop (**primary** | **lowestip**) - specifies how to select the next hop for this interface

Description

Define a set of interfaces. Multiple wildcards, names and addresses may be specified. It is an error if the interface list is empty.

If many interface lists with more than one parameter are present in the configuration file, these parameters are collected at run-time to create the specific parameter list for a given interface. If the same parameter is specified on more than one list, the parameter with the most specific interface is used.

Defaults

none

Context

interfaces statement

Examples

```
interfaces { interface en ex1 disable ; } ;
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

preference on page 39

down preference on page 26

enable on page 27

disable on page 25

passive on page 38

simplex on page 44

reject on page 40

blackhole on page 20

As on page 19

alias primary on page 18

aliases-nexthop on page 15

interfaces

Name

`interfaces`

Syntax

```
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 ] ; ]  
};
```

Parameters

`options`

`interface`

`define`

Description

An interface is the connection between a router and one of its attached networks. A physical interface may be specified by interface name, by IP address, or by domain name. Multiple levels of reference in the configuration language allow interfaces to be identified using wildcard or interface type name. The *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.

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 or ::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 router ID to allow routing to a system based on the router ID that will work if some interfaces are down.

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 router ID 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.

Non-broadcast multi-access (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. For consistency, GateD ensures that there is a route available to each IP interface that is configured and up. Normally this is done by the "ifconfig" command that configures the interface.

To ensure consistency, GateD installs a route in the kernel's FIB 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 ensures that packets originating on this host 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 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 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 field. 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.

Defaults

interfaces configured by the UNIX ifconfig(1) command

Context

global

Examples

```
interfaces { options strictinterfaces scaninterval 10:0 ; } ;
interfaces {
    options aliases-nexthop lowestip ;
    interface all down preference 150;
    interface en2 preference 32 down preference 37 passive simplex AS
        18432 alias primary 7.4.17.76 ;
    define p2p local 176.144.13.44 remote 176.144.12.92 nomulticast;
} ;
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

`export` on page 623

`import` on page 605

`options` on page 36

interface on page 28

define on page 22

multicast | nomulticast

Name

`multicast` | `nomulticast` - explicitly enables/disables multicast on a defined interface

Syntax

`multicast` | `nomulticast`

Parameters

none

Description

explicitly enables/disables multicast on a defined interface. By default, IPIP tunnels are multicast, nouncast interfaces.

Defaults

`multicast`

Context

`interfaces` `define` statement

Examples

```
interfaces {  
    define p2p local 176.144.13.44  
        remote 176.144.13.92  
        nomulticast ;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

`interfaces` on page 30

netmask

Name

netmask - defines the netmask of a logical subnet

Syntax

netmask *mask*

Parameters

mask - the network mask for the subnet of the interface being defined

Description

netmask defines the netmask of a logical subnet. It is an error to specify a netmask address on a point-to-point link.

Defaults

netmask defaults to the host mask for point-to-point interfaces, and cannot be changed. For non-point-to-point, the netmask defaults to that of the existing interface with the same local address, if any.

Context

interfaces **define** statement

Examples

```
interfaces {  
    define subnet local 192.168.13.114  
    netmask 255.255.255.0 ;  
};
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

interfaces on page 30

options

Name

options - specifies interface related global options

Syntax

```
options [ strictinterfaces ]  
      [ scaninterval time ]  
      [ aliases-nexthop ( primary | lowestip ) ];
```

Parameters

strictinterfaces - **strictinterfaces** indicates that it is a fatal error to refer to an interface in the configuration file that is not present when GateD is started and not listed in a define statement. Without **strictinterfaces**, a warning message will be issued, but GateD will continue.

scaninterval *time* - **scaninterval** specifies how often GateD scans the kernel interface list for changes. The default is every 15 seconds on systems without a routing socket, and 60 seconds on systems that pass interface status changes through the routing socket. *time* must be in the range of 15-3600 seconds. Note that GateD will also scan the interface list on receipt of a SIGUSR2. The time may be specified as integer seconds, or mintues:seconds, or hours:minutes:seconds. For example:

30 = 30 seconds

1:30 = 1 minute, 30 seconds, or 90 seconds

1:0:0 = 1 hour (or 3600 seconds)

For operating systems with a routing socket, a *time* of zero disables periodic interface scans. Systems that utilize a routing socket that do not prevent against 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.

aliases-nexthop (primary | lowestip) - **aliases-nexthop** specifies which address GateD will install as the next hop for interface routes. If **primary** is used, the primary interface address (default) will be installed. If **lowestip** is used, the address with the lowest IP address will be installed. This **options** value sets a global parameter that may be overridden for particular interfaces using the **interface** option of the same name. **primary** is the recommended setting.

Description

options enables configuration of three global options related to interfaces. The options are **strictinterfaces**, **scaninterval**, and **aliases-nexthop**.

Defaults

strictinterfaces is disabled. **scaninterval** defaults to 15 seconds on systems without a routing socket and 60 seconds otherwise. **aliases-nexthop** defaults to the primary interface address.

Context

`interfaces` statements

Examples

Example 1

Enable strict interfaces.

```
interfaces { options strictinterfaces ; } ;
```

Example 2

Set scan interval to 5 minutes.

```
interfaces { options scaninterval 300 ; } ;
```

Example 3

Use lowest IP address as the next hop for alias interface routes.

```
interfaces { options aliases-nexthop lowestip ; } ;
```

Example 4

For operating systems with a routing socket, a scan-interval of zero disables periodic interface scans.

```
interfaces {  
    options scaninterval 0 ;  
};
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

`interfaces` on page 30

passive

Name

`passive`

Syntax

`passive ;`

Parameters

none

Description

`passive` prevents GateD from changing the preference of the route to this interface if it is not believed to be functioning properly due to lack of received routing information. A mechanism called aging is implemented by protocol modules to detect a likely malfunctioning interface. Protocols using an interface maintain timestamps indicating when a message was last received over the interface. The timestamps are typically updated in response to received "hello" messages. If a protocol message is not received on the interface within an "aging period" (currently, 300 seconds), the interface is marked down. An interface being marked down causes the "interface route" preference to be changed to "down preference".

Defaults

This "aging mechanism" is enabled only if the GateD configure option "`--enable-interfacer_aging`" is configured. Otherwise, aging is disabled. Also note that "aging" does not take place if no protocols are using the interface.

Context

`interfaces` statement

Examples

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

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

`down preference` on page 26

`interfaces` on page 30

`preference` on page 39

preference

Name

preference - used to select the best route when multiple routes exist for the same destination

Syntax

```
preference preference ;
```

Parameters

preference - Preferences are in the range 0 to 255 inclusive, with 0 being the highest preference a route may have.

Description

Multiple routes (prefix, next hop) may exist for the same destination. When multiple routes exist for the same destination, the route's preference is used to select the best route. For interface routes, **preference** sets the preference for routes to this interface when it is up and appears to be functioning properly.

Defaults

interface routes default to:

```
preference 0;
```

static routes default to:

```
preference 60;
```

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface all preference 14;  
};
```

See Also:

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

"Chapter 19 Static Routes" on page 101 of *Configuring GateD*

down preference on page 26

interface on page 28

interfaces on page 30

reject

Name

reject - causes the local address of this interface to be used as the next hop for reject routes.

Syntax

```
reject ;
```

Parameters

none

Description

reject specifies that this interface is the reject pseudo-interface supported by some older kernels. This interface must be of type "loopback" and should discard any packets sent to it, sending back an "unreachable message". This option is archaic and should no longer be necessary. Modern kernels recognize the RTF_REJECT flag associated with reject routes and directly perform the desired action of discarding packets sent to those routes and returning an "unreachable message" to the originator of the packet.

Defaults

reject is disabled by default.

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface rj0 reject;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

"Chapter 19 Static Routes" on page 101 of *Configuring GateD*

blackhole on page 20

remote

Name

remote - specifies the remote address of a logical point-to-point link

Syntax

remote *address*

Parameters

address - the host address of the remote system on a point-to-point interface

Description

The **remote** *address* specifies the remote address of the logical point-to-point link. It is an error to specify a remote address if **subnet** is specified as the defined interface type.

Defaults

none

Context

interfaces **define** statement

Examples

```
interfaces {  
    define p2p local 176.144.13.44  
        remote 176.144.13.92  
        nomulticast ;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

scaninterval

Name

scaninterval - specifies how often GateD scans the kernel interface list for changes

Syntax

scaninterval *time*

Parameters

time - is a time from 0 to 3600 seconds and can be specified as:

seconds

minutes:seconds

hours:minutes:seconds

Description

scaninterval specifies how often GateD scans the kernel interface list for changes. The default is every 15 seconds on systems that don't have a routing socket, and 60 seconds on systems that pass interface status changes through the routing socket. *time* must be in the range of 15-3600 seconds. Note that GateD will also scan the interface list on receipt of a SIGUSR2. The time may be specified as integer seconds, or minutes:seconds, or hours:minutes:seconds. For example:

30 = 30 seconds

1:30 = 1 minute, 30 seconds, or 90 seconds

1:0:0 = 1 hour (3600 seconds)

For operating systems with a routing socket, a time of zero disables periodic interface scans. Systems that utilize a routing socket that do not prevent against 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.

Defaults

scaninterval 15 for most systems

scaninterval 60 on systems that pass interface status changes through the routing socket

Context

interfaces options statement

Examples

Set scan interval to 5 minutes.

```
interfaces { options scaninterval 5:0 ; } ;
```

See Also

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*
interfaces on page 30

simplex

Name

simplex - defines an interface as unable to hear its own multicast packets

Syntax

```
simplex ;
```

Parameters

none

Description

simplex defines an interface as unable to hear its own multicast packets. Some systems define an interface as simplex with the IFF_SIMPLEX flag. On others, it needs to be specified in the configuration file. On simplex interfaces, a received multicast packet with a source address belonging to the interface on which the packet was received is assumed to have been looped back in software and is not used as an indication that the interface is functioning properly. This flag is valid on non-broadcast interfaces.

Defaults

If the interface has been configured by the utility ifconfig(1), then the default is based on the kernel IFF_SIMPLEX flag; otherwise, the simplex flag is off by default.

Context

```
interfaces interface statement
```

Examples

```
interfaces {  
    interface en simplex;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

strictinterfaces

Name

strictinterfaces - indicates that it is a fatal error to refer to an interface in the configuration file that is not present when GateD is started and not listed in a **define** statement

Syntax

```
strictinterfaces
```

Parameters

none

Description

strictinterfaces indicates that it is a fatal error to refer to an interface in the configuration file that is not present when GateD is started and not listed in a **define** statement. Without **strictinterfaces**, a warning message will be issued, but GateD will continue.

Defaults

strictinterfaces defaults to off.

Context

interfaces options statement

Examples

```
interfaces { options strictinterfaces ; } ;
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

tunnel

Name

tunnel - defines the encapsulation protocol to use for a point-to-point tunnel

Syntax

```
tunnel encapsulation_protocol
```

Parameters

encapsulation_protocol - Currently, the only valid protocol is **ipip**.

Description

tunnel defines the encapsulation protocol to use for a point-to-point tunnel. Currently only IPIP encapsulation (RFC 2003) is recognized. (IPIP encapsulation is used by multicast tunnels supported by kernels.) RFC 2003 is available at:

<http://www.ietf.org/rfc/rfc2003.txt>

Defaults

A defined interface is not a tunnel by default.

Context

interfaces **define** statement

Examples

```
interfaces {  
    define p2p local 198.108.60.89 remote  
    141.213.10.41 multicast nouncast  
    tunnel ipip;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

interfaces on page 30

unicast | nouncast

Name

`unicast` | `nouncast` - explicitly enables/disables unicast on the interface

Syntax

`unicast` | `nouncast`

Parameters

none

Description

explicitly enables/disables unicast on the interface. By default, IPIP tunnels are multicast, nouncast interfaces. Setting this flag causes a route for this interface to not be installed in the kernel.

Defaults

`unicast` is the default for all defined interfaces.

Context

`interfaces` `define` statement

Examples

```
interfaces {  
    define p2p local 198.108.60.89 remote  
    141.213.10.41 multicast nouncast  
    tunnel ipip;  
};
```

See Also

"Chapter 7 Interface Statement" on page 23 of *Configuring GateD*

`interfaces` on page 30

Chapter 7

Routing Information Protocol (RIP)

authentication

Name

authentication - sets the authentication used on an interface

Syntax

```
[ 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 ] ;  
} ; ]
```

Parameters

password - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, "a" or "Whoever#")

start-accept YYYY/MM/DD HH:MM - a time specification for the start accept time for this key

stop-accept YYYY/MM/DD HH:MM - a time specification for the stop accept time for this key

start-generate YYYY/MM/DD HH:MM - a time specification for the start generate time for this key

stop-generate YYYY/MM/DD HH:MM - a time specification for the stop generate time for this key

number - the key ID to be used with this key

Description

The authentication command is used both to verify and to generate the authentication field in the RIP header for all packets sent and received on the specified interface. This applies

only to version 2 packets and RIPv1-compatible RIPv2 packets, because RIPv1 does not support authentication. One exception to the above is the case of query packets. In the case of querying a router through an interface that is speaking RIPv1, the query packet will still be authenticated, but no authentication will be used on the outgoing packet.

If a packet is received with authentication that does not match (including the case where a packet is received with authentication when none is specified on the interface), then it is ignored.

On a trusted network, simple authentication can be used to create two logical networks because sets of routers with shared passwords will talk to each other, but not communicate with those using a different password. On an untrusted network, however, this technique should not be used because simple passwords are sent in clear text. MD5 should be used instead. However, even MD5 does not encrypt the entire packet, only the authentication field of the header.

Specifying anything other than authentication **none** and not specifying **version 2** will cause a parse error. For somewhat more serious authentication, use **trustedgateways** or a combination of **trustedgateways** and **authentication**.

Defaults

```
authentication none ;
```

Context

```
rip interface statement
```

Examples

Example 1

The following command resets the default behavior.

```
rip on {  
    interface all authentication none;  
};
```

Example 2

To set up two virtual RIP networks, one running on interfaces eth0 and eth1 and one running on eth2 and eth3, something like the following can be used.

```
rip on {  
    interface eth0 eth1 version 2  
        authentication simple "foo";  
    interface eth2 eth3 version 2  
        authentication simple "bar";  
};
```

Example 3

On a network where users are not trusted, MD5 authentication can be used. The following example sets MD5 keys for a month, each overlapping for an hour.

```
rip on {
    interface all version 2 authentication md5 {
        key "KeepOut" id 1 {
            start-accept 2000/05/01 00:00;
            stop-accept 2000/05/08 01:00;
        };
        key 192.168.10.1 id 2 {
            start-accept 2000/05/08 00:00;
            stop-accept 2000/05/16 01:00;
        };
        key 0xff id 3 {
            start-accept 2000/05/16 00:00;
            stop-accept 2000/05/23 01:00;
        };
        key "TheEnd" id 4 {
            start-accept 2000/05/23 00:00;
            stop-accept 2000/06/01 01:00;
        };
    };
};
```

Example 4

There is one point of potential confusion in Example 3, in that a semicolon is not necessary after the braces surrounding the MD5 keys, but it is necessary after **interface**. So, the following gives results identical to Example 3.

```
rip on {
    interface all authentication md5 {
        key "KeepOut" id 1 {
            start-accept 2000/05/01 00:00;
            stop-accept 2000/05/08 01:00;
        };
        key 192.168.10.1 id 2 {
            start-accept 2000/05/08 00:00;
            stop-accept 2000/05/16 01:00;
        };
        key 0xff id 3 {
```

```
        start-accept 2000/05/16 00:00;
        stop-accept 2000/05/23 01:00;
    };
    key "TheEnd" id 4 {
        start-accept 2000/05/23 00:00;
        stop-accept 2000/06/01 01:00;
    };
} version 2;
};
```

See Also

`ripin` on page 70
`ripout` on page 72
`metricin` on page 60
`metricout` on page 62
`version` on page 83
`secondary authentication` on page 74
`sourcegateways` on page 76
`trustedgateways` on page 80
`interface` on page 58

broadcast, nobroadcast

Name

broadcast, **nobroadcast** - specifies whether RIP is an active or passive participant

Syntax

```
broadcast ;  
nobroadcast ;
```

Parameters

none

Description

RIP is most often run on a multi-homed box to cause the machine to act as a router in small LANs; however, it is also run on single-homed machines, often to learn the default route or routes to other networks in the domain. RIP will try to guess whether it should act as a router (i.e., be an active participant in routing) by examining the number of interfaces available to it. **[no]broadcast** can be used to override this default behavior. Specifically, if **broadcast** is specified, then RIP updates will be sent even if only one interface is present; conversely, if **nobroadcast** is specified, RIP updates will not be sent, regardless of the number of interfaces present.

[no]broadcast is most often used in its **nobroadcast** form to turn off active participation in a RIP cloud. This can be used, for example, on a multi-homed machine running different protocols on different interfaces where it is desired not to send the other attached networks or learned routes into the RIP cloud. The same functionality can be achieved through policy rules. But for a simple network, that degree of configuration complexity is not necessary.

Either version of **[no]broadcast** applies only to broadcasting (or multicasting, in the case of RIP version 2) of RIP packets. Therefore, it will lack effect if **sourcegateways** is present.

broadcast should not be confused with the broadcast argument to **version**.

Defaults

RIP will default to broadcasting if configured **on** and if there is more than one attached interface. It will default to only listening if there is just one attached interface.

Context

rip statement

Examples

Example 1

The following example configures RIP to broadcast updates, even on a single-homed machine. This might be useful for exporting static routes that are configured only on the

single-homed machine; however, in some cases, use of broadcast on a single-homed machine can result in data packets traversing a single network twice.

```
rip on {  
    broadcast;  
};
```

Example 2

The following example turns broadcasting off, even on a multi-homed machine.

```
rip on {  
    nobroadcast;  
};
```

See Also

`export` on page 623

`import` on page 605

`sourcegateways` on page 76

defaultmetric

Name

defaultmetric - sets the default metric for advertising RIP routes learned from other protocols

Syntax

```
defaultmetric defmetnum;
```

Parameters

defmetnum - a RIP metric from 1 to 16, inclusive

Description

When RIP learns a route from another protocol, the metrics of the two protocols are almost certainly incompatible. Most protocols have a much larger metric space (for example, 0 to 65535) than RIP does, and, therefore, a mechanism is required for specifying the initial metric that the RIP advertiser will use.

defaultmetric is used to set the default metric for advertising into the RIP cloud routes learned from other protocols, including **static**. Use the **export** statement to set this metric for exporting routes to RIP on a per-protocol basis.

Defaults

```
defaultmetric 1;
```

Context

rip statement

Examples

The following example sets the default metric for routes exported to RIP to be 2.

```
rip on {  
    defaultmetric 2;  
};
```

See Also

export page 623

expire-time

Name

expire-time - sets the expiration time for routes received via RIP

Syntax

```
expire-time expire_time;
```

Parameters

expire_time - an integer from 5 to 180, inclusive

Description

expire-time is initialized when a route is established, and any time an update message is received for the route. If *expire_time* seconds elapse from the last time the **expire-time** was initialized, the route is considered to have expired, and the deletion process begins for that route. This is a **group** option in the **rip** clause. For example, it is used in the same context as **preference** and **broadcast**.

Defaults

```
expire-time 180;
```

Context

rip statement

Examples

```
rip yes {  
    expire-time 100;  
    interface fxp0;  
};
```

See Also

update-time on page 82

ignorehostroutes

Name

`ignorehostroutes` - ignores host routes learned in route responses

Syntax

```
ignorehostroutes;
```

Parameters

none

Description

Sometimes it is desirable to ignore host routes (routes with a netmask of 255.255.255.255) advertised by other routes.

Note: This is effective only with RIP version 2. (RIP version 1 does not send netmask information in its updates.)

Context

`rip` statement

Examples

```
rip on {  
    interface eth0 version 2;  
    ignorehostroutes;  
};
```

See Also

`version` on page 83

`interface` on page 58

interface

Name

interface - specifies interface(s) on which RIP operates

Syntax

```
interface interfacelist [ripIFcmd [ripIFcmd ... ] ];
```

Parameters

interfacelist - the list of interfaces on which RIP will operate with the given *ripIFcmd*
mdS

Description

interface is used both to set interface-specific parameters in RIP and to determine on which interfaces RIP will be run. By default, RIP will run on all interfaces; however, if any specific interfaces (or subsets of interfaces) are explicitly configured with **interface**, then all non-configured interfaces will not run RIP.

If multiple interfaces (either physical or logical) have addresses on the same subnet, then (regardless of configuration) RIP will send updates only on the first one for which RIP is configured to do so.

Although it is possible to specify a loopback interface or loopback address in an interface statement, RIP will not normally send packets to a loopback. To override this behavior, use **sourcegateways** with the loopback address included in the *gatewaylist*.

Defaults

```
interface all;
```

Context

rip statement

Examples

Example 1

The following will configure RIP to run on the eth0 interface only.

```
rip on {  
    interface eth0;  
};
```

Example 2

Alternatively, the following example will set RIP to run on all interfaces except eth0.

```
rip on {  
    interface all;
```

```
        interface eth0 noripin noripout;  
    };
```

See Also

`ripin` on page 70

`ripout` on page 72

`metricin` on page 60

`metricout` on page 62

`version` on page 83

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

metricin

Name

metricin - sets an additional metric on incoming RIP routes

Syntax

metricin *ripmetric*

Parameters

ripmetric - a number signifying hop count, from 0 to 15

Description

It is often the case that a router should prefer routes received on one set of interfaces over those received on another. For example, given two point-to-point links, one can be more expensive than the other and should, therefore, be less preferred. **metricin** is used for exactly this purpose: to make routes learned from certain interfaces less preferable.

metricin is the default manner by which RIP increments hop count. That is to say, RIP works by adding a hop every time a route is received and before it is sent back out to other interfaces. This implementation adds the hop when the route is received (for example, before decisions regarding whether the route should be used are made). By default, a metric of 1 plus the kernel interface metric is added as the hop count. Normally, this interface metric is zero, but some operating systems allow it to be specified on interface configuration. If **metricin** is explicitly given, it is added as an absolute value (for example, without the interface metric).

Defaults

If left unspecified, a metric of 1 plus the kernel interface metric (if any) is the default.

Context

rip interface statement

Examples

Example 1

To override any specified interface metric, the following adds exactly 1 to the metric of all received routes.

```
rip on {  
    interface all metricin 1;  
};
```

Example 2

In somewhat more normal usage, the following example increases the cost of routes by 2 that are received over a point-to-point link more than those received on other interfaces.

```
rip on {  
    interface all metricin 1;  
    interface ppp0 metricin 3;  
};
```

See Also

`ripin` on page 70
`ripout` on page 72
`metricout` on page 62
`version` on page 83
`authentication` on page 49
`secondary authentication` on page 74
`sourcegateways` on page 76
`interface` on page 58

metricout

Name

metricout - specifies an additional cost to be added to outgoing routes

Syntax

metricout *ripmetric*

Parameters

ripmetric - a number signifying hop count, from 0 to 15

Description

Normally, this RIP implementation adds to the hop count only on incoming routes. There are times, however, when the user wants to cause other routers not to prefer routes from a given origin. For example, if the router is a backup router, it might be desirable for its routes to always be less preferred. **metricout** accomplishes this by adding to the RIP metric on top of any metric specified by **metricin** before RIP updates are sent out the specified interface.

Defaults

metricout 0 ;

Context

rip interface statement

Examples

Example 1

If this router (host 128) is acting as backup on 192.168.10/24 for all other interfaces, the following will add 2 to all metrics advertised out the 192.168.10.128 interface.

```
rip on {  
    interface all ripin ripout;  
    interface 192.168.10.128 metricout 2;  
};
```

Example 2

The following example has no effect. It is contrived to point out that **metricout** does not impact routing based on the receipt of updates, so if updates do not go out an interface on which **metricout** is specified, the behavior of the entire network will be identical to what it would have been without **metricout** being issued.

```
rip on {  
    interface all ripin ripout;  
    interface eth0 ripin noripout metricout 15;
```

`};`

See Also

`ripin` on page 70
`ripout` on page 72
`metricin` on page 60
`version` on page 83
`authentication` on page 49
`secondary authentication` on page 74
`sourcegateways` on page 76
`interface` on page 58

nocheckzero

Name

nocheckzero - specifies handling of zero-filled reserved fields in RIP version 1

Syntax

nocheckzero ;

Parameters

none

Description

The RIP version 1 specification mandates that certain fields in RIP routing updates are reserved and should be filled with zeros. It also mandates that version 1 packets, where the reserved fields are not filled with all zeros, should be discarded. However, some implementations of RIP do not follow the specification and carry non-zero information in these fields. **nocheckzero** allows RIP to interoperate with those broken implementations.

The user should be certain that the noncompliant router is intentionally generating malformed packets (rather than malfunctioning) before enabling interoperability. Otherwise, correct routing could be compromised.

Defaults

Zero-filled reserved fields are checked for RIP version 1.

Context

rip statement

Examples

The following example enables interoperability with a router that is not filling reserved fields with zeros:

```
rip on {  
    nocheckzero;  
};
```

preference

Name

preference - sets the preference for RIP routes

Syntax

preference *prefvalue* ;

Parameters

prefvalue - number from 0 to 255

Description

Routers can learn multiple routes to the same destination. Each routing protocol handles this internally, so, for example, if RIP learns two different routes to the same destination, it will select the route with the lower metric. However, a router can learn two routes to the same destination from different routing protocols, and it must find a way to choose between them. GateD uses a preference number to make this decision, choosing the route with the lower preference. Normally, RIP routes are preferred to routes from external routing protocols (for example, BGP or OSPF ASE) and to those that are generated or aggregated. RIP routes are less preferable than those learned from other intra-domain routing protocols and those that are statically configured. **preference** can be used to change this ordering.

Defaults

preference 100;

Context

rip statement

Examples

Example 1

The following example makes RIP routes more preferable than IS-IS routes, but less preferable than OSPF routes.

```
rip on {  
    preference 13;  
};
```

Example 2

The following example makes RIP routes less preferable than BGP routes, but still more preferable than EGP routes.

```
rip on {  
    preference 180;
```

```
};
```

See Also

"Preferences and Route Selection" on page 11 of *Configuring GateD*

"Route Importation" on page 137 of *Configuring GateD*

`import` on page 605

query authentication

Name

query authentication - sets the authentication used by the **ripquery** utility

Syntax

```
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 ] ;
} ; ]
```

Parameters

password - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, "a" or "Whoever#")

start-accept *YYYY/MM/DD HH:MM* - a time specification for the start accept time for this key

stop-accept *YYYY/MM/DD HH:MM* - a time specification for the stop accept time for this key

start-generate *YYYY/MM/DD HH:MM* - a time specification for the start generate time for this key

stop-generate *YYYY/MM/DD HH:MM* - a time specification for the stop generate time for this key

number - the key ID to be used with this key

Description

Certain utilities, such as **ripquery**, have been created for debugging RIP routers. These tools send a RIP POLL packet, which is an extension undocumented in the RFC.

query authentication is used to check the incoming POLL packets. If the authentication matches, then RIP will reply with its full routing table. (For example, it will not run split-horizon or poison reverse before replying.) If the authentication does not match, then the request will be discarded.

Although **query authentication** uses the standard key format for passwords, the generate portions of the key are irrelevant because the key is used only to check incoming requests. Outgoing packets use whatever authentication method is set up on the interface over which the POLL packet was received.

Defaults

`query authentication none;`

Context

`rip` statement

Examples

The following example sets a simple password for authentication of RIP POLL packets:

```
rip on {  
    query authentication simple 0xdeadbeef;  
};
```

See Also

`authentication` on page 49

`secondary authentication` on page 74

`"ripquery"` on page 17 of *Operating GateD*

rip

Name

rip - configures the Routing Information Protocol (RIP)

Syntax

```
rip on ;  
rip on { ripargs } ;  
rip off ;
```

Parameters

ripargs - RIP configuration parameters, which are described throughout this chapter

Description

One of the most widely used interior gateway protocols is RIP. RIP is an implementation of a distance-vector routing protocol (using the Bellman-Ford algorithm) for local networks. RIP classifies routers as active or 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.

Defaults

The default for RIP is **off** unless the **--enable-ripon** flag is passed to **configure**.

Context

global

Examples

The following example configures RIP to run on all interfaces.

```
rip on;
```

See Also

bgp on page 232

ospf on page 124

isis on page 185

dvmrp on page 364

import on page 605

export on page 623

aggregate on page 673

dampen-flap on page 707

“Routing Information Protocol” on page 39 of *Configuring GateD*

ripin, noripin

Name

ripin, **noripin** - specifies whether RIP will listen to RIP updates

Syntax

ripin
noripin

Parameters

none

Description

ripin specifies that RIP will process RIP updates received on a given interface. Since this is also the default, it is not normally required, except to override a **noripin** specified on a wildcard list of interfaces. **noripin** does exactly the opposite. Although it would almost certainly be a misconfiguration, it is important to note that RIP can send RIP updates on a superset of those interfaces on which it receives updates. This can be a valid configuration if, for example, the user receives RIP updates from an ISP, redistributing those onto the LAN, and does not want to send the LAN topology back to the ISP. But this would be highly unusual.

Defaults

ripin

Context

rip interface statement

Examples

This somewhat contrived example processes RIP updates received on all eth devices except eth0.

```
rip on {  
    interface eth ripin;  
    interface eth0 noripin;  
};
```

See Also

ripout on page 72

metricin on page 60

metricout on page 62

version on page 83

authentication on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

ripout, noripout

Name

ripout, **noripout** - specifies whether RIP will send RIP updates

Syntax

ripout
noripout

Parameters

none

Description

noripout specifies that RIP updates should not be sent on the specified interfaces. **ripout** specifies the converse and is the default on broadcast-enabled interfaces.

Defaults

ripout on broadcast interfaces

noripout on nonbroadcast interfaces (including point-to-point interfaces)

Context

rip interface statement

Examples

Example 1

The following example specifies that RIP updates should not be sent on any eth interfaces except eth0.

```
rip on {  
    interface eth noripout;  
    interface eth0 ripout;  
};
```

Example 2

The following example specifies that RIP updates should be sent on all eth and ppp interfaces.

```
rip on {  
    interface eth ripout;  
    interface ppp ripout;  
};
```

See Also

- `ripin` on page 70
- `metricin` on page 60
- `metricout` on page 62
- `version` on page 83
- `authentication` on page 49
- `secondary authentication` on page 74
- `sourcegateways` on page 76
- `interface` on page 58

secondary authentication

Name

secondary authentication - sets the authentication used on an interface

Syntax

```
[ 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 ] ;
} ; ]
```

Parameters

password - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, "a" or "Whoever#")

start-accept YYYY/MM/DD HH:MM - a time specification for the start accept time for this key

stop-accept YYYY/MM/DD HH:MM - a time specification for the stop accept time for this key

start-generate YYYY/MM/DD HH:MM - a time specification for the start generate time for this key

stop-generate YYYY/MM/DD HH:MM - a time specification for the stop generate time for this key

number - the key ID to be used with this key

Description

secondary authentication is identical in functionality to **authentication**, with the exception that it is used only for checking authentication and is used only if the primary authentication method fails.

secondary authentication can be used while a network is in transition to make sure that old passwords are still accepted.

Defaults

secondary authentication none ;

Context

rip interface statement

Examples

In order to transition a network from simple password "foo" to simple password "bar," something like the following can be used until all routers are updated to the new password.

```
rip on {  
    interface all version 2 authentication simple "foo"  
        secondary authentication simple "bar";  
};
```

See Also

- `ripin` on page 70
- `ripout` on page 72
- `metricin` on page 60
- `metricout` on page 62
- `version` on page 83
- `authentication` on page 49
- `sourcegateways` on page 76
- `trustedgateways` on page 80
- `interface` on page 58

sourcegateways

Name

sourcegateways - sets routers to which RIP will send updates

Syntax

```
sourcegateways gatewaylist;
```

Parameters

gatewaylist - a comma-separated list of IP addresses or host names

Description

sourcegateways provides a complete list of IP addresses to which the configured router will unicast RIP updates.

RIP must be configured on an interface that shares a subnet with each source gateway.

If any source gateways are specified, then no RIP packets will be broadcast or multicast. They will only be unicast to the list of source gateways. As a result, the arguments to **version** do not exactly match their denotation. If version 1 is specified, v1 packets will be unicast to the list of sourcegateways. If version 2 is specified, and the argument to it is broadcast, v1 compatible, then v2 packets will be unicast. If version 2 is specified and the argument to it is multicast, then v2 packets will be unicast to the list of sourcegateways. If no sourcegateways are specified, GateD will not unicast updates.

Defaults

By default, RIP will either broadcast or multicast its updates.

Context

rip statement

Examples

Example 1

This example sets a RIP router to send updates to routers with only host addresses of 2 on the 192.168.10 and .12 networks.

```
rip on {  
    interface eth;  
    sourcegateways 192.168.10.2, 192.168.12.2;  
};
```

Example 2

Similarly, the following example shows how this can be accomplished with host names (but DNS must be relied upon).

```
rip on {  
    interface eth;  
    trustedgateways rtr1.my.domain, rtr2.my.domain;  
};
```

See Also

`version` on page 83

`interface` on page 58

traceoptions

Name

traceoptions - sets RIP-specific tracing options

Syntax

```
traceoptions trace_options ;
```

Parameters

trace_options - In addition to the trace options specified in "Chapter 4 Trace Statements" on page 15, the following trace options are available for the RIP protocol:

request - Trace RIP information request packets, which include request, poll, and poll entry packets.

response - Trace RIP response packets (for example, those that actually contain routing updates).

other - Trace any other type of RIP packet.

packets - Trace all of the above.

or one of these packet tracing options, optionally prepended by one of the three modifiers (**send**, **recv**, **detail**):

Description

RIP tracing options behave similarly to all other tracing options. The RIP-specific packet tracing options available include: **request**, **response**, **other**, and **packets**.

Each of the above can be modified by **detail** (include more details than normal tracing), **send** (trace only those packets that this router sends), or **recv** (trace only those packets that this router receives).

In addition, the trace option of **policy** will trace whenever a new route is announced, when a metric being announced is changed, or when a route enters or exits holddown. The trace option of **all** traces all of the packets. **none** traces nothing.

Defaults

```
traceoptions none;
```

Context

rip statement

Examples

Example 1

This example saves detail on all packets sent or received to the file `/var/tmp/rip_pkts`.

```
rip on {  
    traceoptions /var/tmp/rip_pkts detail packets;
```

```
};
```

Example 2

This example traces all policy changes.

```
rip on {  
    traceoptions policy;  
};
```

Example 3

This example restores default behavior.

```
rip on {  
    traceoptions none;  
};
```

See Also

`traceoptions` page 3

"Trace Statements" on page 15 of *Configuring GateD*

trustedgateways

Name

trustedgateways - sets routers from which RIP will accept routes

Syntax

```
trustedgateways gatewaylist;
```

Parameters

gatewaylist - a comma-separated list of IP addresses or host names

Description

trustedgateways allows for additional security beyond that provided by **authentication**. Specifically, it provides a complete list of IP addresses from which the configured router will accept RIP updates. Of course, it is still possible to spoof IP addresses, but, short of that, only those RIP packets originating from the listed hosts will be accepted.

Defaults

All routers are trusted.

Context

rip statement

Examples

Example 1

This example sets a RIP router to heed updates sent from routers with only host addresses of 2 on the 192.168.10 and .12 networks.

```
rip on {  
    interface eth;  
    trustedgateways 192.168.10.2, 192.168.12.2;  
};
```

Example 2

Similarly, this can be accomplished with host names (of course, in this situation, the benefit may not be achieved if DNS is not secure).

```
rip on {  
    interface eth;  
    trustedgateways rtr1.my.domain, rtr2.my.domain;  
};
```

See Also

`interface` on page 58

`authentication` on page 49

update-time

Name

update-time - sets the update time for unsolicited route response

Syntax

```
update-time update_time;
```

Parameters

update_time - an integer from 1 to 30, inclusive

Description

This is a group option in the RIP clause. For example, it is used in the same context as **preference** and **broadcast**.

Defaults

```
update-time 30;
```

Context

rip statement

Examples

```
rip yes {  
    update-time 20;  
    interface fxp0;  
};
```

See Also

expire-time on page 56

version

Name

version - specifies the version of RIP to be run

Syntax

version *ripversion* [*ripversarg*]

Parameters

ripversion - 1 or 2

ripversarg - There are no available arguments for version 1; for version 2, valid arguments are **broadcast** and **multicast**.

Description

version is used to override the default version of RIP that will be run on a given interface. Normally, RIPv1 will be run on all interfaces. If version 2 is specified, then the default behavior depends on the capabilities of the interface. If the interface is multicast capable, then RIP updates will be multicast to RIP2-ROUTERS.MCAST.NET (the reserved, multicast address 224.0.0.9). If the interface is not multicast capable, then RIP version 1 compatible version 2 packets will be broadcast.

The *ripversarg* (which, if specified, is exclusively either **broadcast** or **multicast**, and which can only be specified for version 2) allows the above behavior to be overridden. This is normally used to specify that only version 1-compatible packets should be sent for interoperability purposes, even though a given interface is multicast capable.

Another exception is the case in which version 2 multicast is specified on an interface that is not multicast capable (e.g., a point-to-point link). In this case, if **sourcegateways** is also specified, the full RIPv2 packets will be directly unicast to the source gateway on the specified interface.

It is important not to confuse the **broadcast** argument with **version**, which specifies interoperability between RIPv1 and RIPv2, and the **broadcast** command.

Defaults

version 1

version 2 defaults to **version 2 multicast** (for multicast capable interfaces; broadcast otherwise).

Context

rip interface statement

Examples

Example 1

The following example runs RIPv2 on all interfaces, except for eth0. This would be useful if it were necessary to interoperate with old RIP implementations on a certain link in the network.

```
rip on {  
    interface all version 2 multicast;  
    interface eth0 version 2 broadcast;  
};
```

Example 2

This example runs RIPv1 on all interfaces, which is the default behavior.

```
rip on {  
    interface all version 1;  
};
```

Example 3

This example runs full RIPv2 on all multicast capable interfaces.

```
rip on {  
    interface all version 2;  
};
```

Example 4

This example is identical to Example 3 but requires typing an additional word.

```
rip on {  
    interface all version 2 multicast;  
};
```

Example 5

This example specifies that RIPv2 be run in RIPv1-compatible mode on all interfaces, except across a point-to-point link, where full RIPv2 is run. This might be useful if a serial link connected two networks: one that ran RIPv2, and one that ran RIPv1.

```
rip on {  
    interface all version 2 broadcast;  
    interface ppp0 version 2 multicast;  
    sourcegateways the.remote.host;  
};
```

See Also

`ripin` on page 70

`ripout` on page 72
`metricin` on page 60
`metricout` on page 62
`authentication` on page 49
`secondary authentication` on page 74
`sourcegateways` on page 76
`interface` on page 58
`broadcast` on page 53

Chapter 6

Definitions

autonomoussystem

Name

autonomoussystem - sets the autonomous system (AS) number of this router

Syntax

```
autonomoussystem autonomous_system [ loops number ] ;
```

Parameters

autonomous_system - the AS number of this router

loops *number* - is only for protocols supporting AS paths, such as BGP. **loops** controls the number of times this autonomous system may appear in an AS path. *number* is an integer in the range 1 to 10, inclusive. **loops** should not be used in normal operations.

Description

autonomoussystem sets the autonomous system 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.

Defaults

There is no default for *autonomous_system*.

loops 1;

Context

global definition statement

Examples

```
autonomoussystem 7476 loops 2;
```

See Also

“Chapter 8 Definition Statements” on page 29 of *Configuring GateD*

confed-id on page 89

interfaces AS on page 19

confed-id

Name

confed-id - the BGP confederation ID for this router

Syntax

confed-id *confederation_number*

Parameters

confederation_number - the autonomous system (AS) number that this router will present to peers outside of this BGP confederation

Description

A BGP router can be configured to be a member of a BGP confederation where the autonomous system is subdivided into several confederation ASs. When configured as a confederation member using the **confed** keyword in BGP **group** and **peer** statements, this router will represent itself as the configured **autonomoussystem** number to confederation peers and as the configured **confed-id** to non-confederation peers.

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.

Defaults

none

Context

global definition statement

Examples

```
confed-id 65412;
```

See Also

autonomoussystem on page 87

"Chapter 8 Definition Statements" on page 29 of *Configuring GateD*

martians

Name

martians - allows additions to the list of martian addresses

Syntax

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

Parameters

allow - The **allow** parameter can be specified to explicitly allow a subset of a range that was disallowed.

[**inet** | **inet6**] **default** - **default** is equivalent to 0.0.0.0/0 if preceded by **inet**, or ::/0 if preceded by **inet6**. If neither **inet** nor **inet6** is specified, it specifies both 0.0.0.0/0 and ::/0.

host - specifies a host number or address that is to be non-routable or routable (if **allow** is set)

mask - a mask specifying a subnet (for example, ffff::)

network - the network address of the subnet

number - the number of bits in the **mask** *mask*. For example, /32 is equivalent to 255.255.255.255 for an IPv4 address or ffff:ffff:: for an IPv6 address.

exact - specifies that the prefix address and mask must match exactly.

refines - specifies that the prefix address must match up to masklen and prefix mask must be longer than configured mask

between *lower* **and** *upper* - specifies that the prefix address must match up to masklen and prefix masklen must be greater than or equal to *lower* and less than or equal to *upper*.

Description

Martians are networks that are considered illegal to be routed on the Internet. **martians** allows additions to the list of martian addresses. See Chapter 26, "Route Filtering," on page 129 of *Configuring GateD* for more information on specifying ranges. The **allow** parameter may also be specified to explicitly allow a subset of a range that was disallowed. **martians** can also be used for route filtering.

RFC 1918 specifies these networks as 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 between autonomous systems. However, these prefixes can be routed within autonomous systems. 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 list of martians is:

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.

::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

Defaults

```
martians {
    127.0.0.0      mask 255.0.0.0 ;
    240.0.0.0      mask 240.0.0.0 ;
    ::1/128 ;
    fe80::/10 ;
    ff00::/8 ;
    ::0000:127.0.0.0/104 ;
    ::ffff::127.0.0.0/104 ;
    ::0000:240.0.0.0/100 ;
    ::ffff:240.0.0.0/100 ;
};
```

Context

global definition statement

Examples

```
martians {
    host 192.168.14.15 allow ;
    3ffd:ffff:ffff:1::/64
} ;
```

See Also

"Chapter 8 Definition Statements" on page 29 of *Configuring GateD*

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*

routerid

Name

routerid - sets the router identifier for use by the BGP and OSPF protocols

Syntax

routerid *host* ;

Parameters

host - specifies an interface address to be used as the router's router ID. The address must be present as a local address on an interface.

Description

routerid sets the router identifier for use by the BGP and OSPF protocols.

Defaults

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.

Context

global definition statement

Examples

```
routerid 32.56.34.89;
```

See Also

"Chapter 8 Definition Statements" on page 29 of *Configuring GateD*

interfaces statement on page 30

Chapter 8

Open Shortest Path First Protocol (OSPF)

advertise-subnet

Name

advertise-subnet - specifies whether OSPF will, when advertising point-to-point interfaces, advertise the network number and netmask of the point-to-point interface instead of a host-route to the remote IP

Syntax

```
advertise-subnet on | off ;
```

Parameters

on | off - enables/disables the option

Description

advertise-subnet specifies whether OSPF will, when advertising point-to-point interfaces, advertise the network number and netmask of the point-to-point interface instead of a host-route to the remote IP. Because sometimes the netmask is set improperly on point-to-point interfaces, the default is **off**. The global **advertise-subnet** will be the default unless it is overridden in the area or interface statement.

Default

```
advertise-subnet off ;
```

Context

ospf statement

area statement

interface statement

virtuallink statement

Examples

```
ospf yes {  
    advertise-subnet on;
```

```
    area 1.2.3.4 {  
        interface ppp0 cost 1;  
    };  
};
```

See Also

`interface` on page 110

always-update-summary

Name

`always-update-summary` - forces GateD to update summary LSAs

Syntax

`always-update-summary on | off ;`

Parameters

`on` | `off` - enables/disables this option

Description

If this option is set, summary LSAs whose supporting path type has changed are always regenerated in the SPF computation. This occurs regardless of whether the contents of the LSAs have changed. This flag is provided to disable an optimization that can cause GateD to fail commercially available, component-level tests.

Default

`on`

Context

`ospf` statement

Examples

```
ospf yes {
    always-update-summary on;
    area 1.2.3.4 {
        interface ppp0 cost 1;
    };
};
```

See Also

`ospf` on page 124

area

Name

area - defines an OSPF area

Syntax

area *areanumber*

Parameters

areanumber - dotted-quad area-ID for this area

Description

Each OSPF router must be configured into at least one OSPF area. If more than one area is configured, at least one must be the backbone. The backbone interface can be a virtual link.

Default

The default area configuration (if no areas are configured by the user) is a backbone with all interfaces.

Context

ospf statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

interface on page 110

backbone on page 101

auth

Name

auth - specifies the type of authentication and key values

Syntax

```
auth [ none | simple auth_key | md5 md5-keyset ]
```

Parameters

none - specifies no authentication

simple - specifies simple (clear password) authentication

md5 - specifies md5 cryptographic authentication

Description

auth is used by OSPF authentication to generate and verify the authentication field in the OSPF header. The global authentication will be the default unless it is specified in the area or interface statement. *auth_key* is specified by one to eight decimal digits (with a value between 0 and 255) separated by periods, a 1- to 8-byte hexadecimal string preceded by 0x, or a one- to eight- character string in double quotes. See "Authentication" on page 99 of *Configuring GateD* for a more detailed description.

Specify MD5 authentication with the *md5-keyset*, which is specified as:

```
key md5-key id id-number [ {  
    [ start-generate date-time; ]  
    [ stop-generate date-time; ]  
    [ start-accept date-time; ]  
    [ stop-accept date-time; ]  
} ];
```

where *md5-keyset* is a one- to 16-character string in double quotes, *id-number* is an integer with a value 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.

Default

```
auth none ;
```

Context

ospf statement

area statement

ospf interface statement

virtuallink statement

Examples

Example 1

```
ospf yes {
    auth simple "foobar";
    backbone {
        interface fxp0 cost 1;
    };
};

ospf yes {
    area 1.2.3.4 {
        auth simple "foo";
        interface fxp1 cost 1;
    };
    backbone {
        auth simple "bar";
        interface fxp2 cost 1;
    };
};
```

Example 2

```
ospf yes {
    traceoptions all;
    backbone {
        interface 192.168.32.17 cost 1
        {
            retransmitinterval 5;
            transitdelay 1;
            priority 1;
            hellointerval 10;
            routerdeadinterval 40;
            auth md5 key "key1" id 123 { stop-accept 2000/01/01 12:00; };
        };
    };
};
```

See Also

[interface](#) on page 110

[area](#) on page 98

backbone

Name

backbone - defines an OSPF backbone area

Syntax

backbone

Parameters

backbone - defines the backbone area

Description

Each OSPF router must be configured into at least one OSPF area. If more than one area is configured, at least one must be the backbone. The backbone interface may be a virtual link. Note that **area 0.0.0.0** or **area 0** is equivalent to **backbone**.

Default

The default area configuration (if no areas are configured by the user) is a backbone with all interfaces.

Context

ospf statement

Examples

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

See Also

interface on page 110

area on page 98

cost

Name

cost - specifies the default cost when importing non-OSPF routes into OSPFASE

Syntax

```
cost defasecost;
```

Parameters

defasecost - default cost of routes exported into Autonomous System External (ASE)

Description

cost is used when exporting a non-OSPF route from the GateD routing table into OSPF as an ASE route. **cost** can be explicitly overridden in export policy.

Default

```
cost 1 ;
```

Context

```
ospf defaults statement
```

Examples

```
ospf yes {  
    defaults {  
        cost 1;  
    };  
    area 1.2.3.4 {  
        interface fxp0;  
    };  
};
```

See Also

defaults on page 103

nssa-cost on page 117

defaults

Name

defaults - specifies the defaults used when importing OSPF ASE routes or Not So Stubby Area (NSSA) routes into the GateD routing table and when exporting routes from the GateD routing table into OSPF ASE or NSSA

Syntax

defaults

Parameters

preference *defasepref*
nssa-preference *defnssapref*
cost *defasecost*
nssa-cost *defnssacost*
tag [**as**] *tagvalue*
type 1 | 2
nssa-type 1 | 2
inherit-metric
nssa-inherit-metric
ribs **unicast** [**multicast**]

Description

These parameters specify the defaults used when importing OSPF ASE routes or NSSA routes into the GateD routing table and exporting routes from the GateD routing table into OSPF ASEs or NSSAs.

Default

The individual options have the following default values:

```
preference 150;
nssa-preference 150;
cost 1;
nssa-cost 1;
tag as 0 ;
type 2;
nssa-type 2;
ribs unicast;
```

Context

ospf statement

Examples

```
ospf yes {
    defaults {
        preference 10;
        cost 1;
        tag 2112;
        type 1;
        inherit-metric;
        ribs unicast multicast;
    };
    area 1.2.3.4 {
        interface fxp0 cost 1;
    };
};
```

See Also

`preference` on page 127
`cost` on page 102
`tag` on page 145
`type` on page 150
`inherit-metric` on page 109
`ribs` on page 132
`area` on page 98
`nssa-cost` on page 117
`nssa-inherit-metric` on page 118
`nssa-preference` on page 119
`nssa-type` on page 120
`nssa` on page 116

disable

Name

`disable` - disable the interface

Syntax

```
disable;
```

Parameters

none

Description

This option causes interface(s) matching the policy to be disabled.

Default

This option is disabled by default.

Context

`ospf interface` statement

Examples

```
ospf yes {  
    interface fxp cost 1;  
    interface fxp0 disable;  
};
```

See Also

`ospf interface` on page 110

`enable` on page 106

enable

Name

`enable` - enable the interface

Syntax

```
enable;
```

Parameters

none

Description

This option is provided for symmetry with 'disable'. It is the default.

Default

This option is enabled by default.

Context

`ospf interface` statement

Examples

```
ospf yes {  
    interface fxp0 { enable };  
};
```

See Also

`ospf interface` on page 110

`disable` on page 105

hellointerval

Name

hellointerval - the length of time, in seconds, between hello packets that the router sends on an interface

Syntax

```
hellointerval time ;
```

Parameters

time - the default length of time to use between sending hello packets

Description

hellointerval is the length of time, in seconds, between hello packets that the router sends on the interface. This option can be specified at the global level and overridden at the area and interface levels.

Default

```
hellointerval 10 ;
```

Context

ospf statement

area statement

interface statement

virtuallink statement

Examples

```
ospf yes {
    hellointerval 20;
    area 1.2.3.4 {
        interface fxp0 cost 1;
    };
    backbone {
        hellointerval 10;
        interface fxp1 cost 2;
    };
    area 2.3.4.5 {
        hellointerval 30;
        interface fxp2;
    };
};
```

```
};
```

See Also

`routerdeadinterval` on page 133

`interface` on page 110

inherit-metric

Name

inherit-metric - configures an OSPF ASE route to inherit the metric of the external route when no metric is specified on the export policy

Syntax

```
inherit-metric ;
```

Parameters

none

Description

inherit-metric allows an OSPF ASE route to inherit the metric of the external route when no metric is specified on the export policy. **inherit-metric** maintains compatibility with all the current export functions. A metric specified on the export policy will take precedence. The **cost** specified in the **default** statement (*defasecost*) will be used if **inherit-metric** is not specified.

Default

The default is to not inherit the external metric.

Context

ospf defaults statement

Examples

```
ospf yes {
    defaults {
        inherit-metric;
    };
    backbone {
        interface fxp0 cost 1;
    };
};
```

See Also

defaults on page 103

cost on page 102

nssa-inherit-metric on page 118

interface

Name

interface - configures an interface to run OSPF

Syntax

```
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] ;
} ] ;
```

Parameters

retransmitinterval *time*

transitdelay *time*

priority *priority*

```

hellointerval time
routerdeadinterval time
pollinterval time
passive
advertise-subnet on | off
auth [ none | simple auth_key | md5 md5-key ]
interface_name - name of interface to use
interface_address - logical address of interface to use
strict-routers on | off - nonbroadcast and point-to-multipoint only
routers - nonbroadcast and point-to-multipoint only
gateway_list - nonbroadcast and point-to-multipoint only

```

Description

This form of the **interface** clause is used to configure an interface in OSPF. Each interface has a cost. The costs of all interfaces that a packet must cross to reach a destination are summed to get the cost to that destination. The default cost is 1, but another non-zero value may be specified.

The **nonbroadcast** form of the **interface** clause is used to specify a nonbroadcast interface on an NBMA medium. Because an OSPF broadcast medium must support IP multicasting, a broadcast-capable medium that does not support IP multicasting must be configured as a nonbroadcast interface. This includes the loopback interface on many operating systems.

The **point-to-multipoint** form of the **interface** clause is used to specify a point-to-multipoint interface. This form can be used when the network does not provide full connectivity to all routers on the network.

Both nonbroadcast and point-to-multipoint require a router's section to specify the routers with which to exchange hellos. This is because they cannot use multicast to locate neighbors.

Default

If an area is configured, the default is no interfaces enabled. When no areas are configured, the default configuration is a backbone area with all interfaces enabled.

For the nonbroadcast form, the default mode is broadcast/point-to-point. The default cost is 1.

For the point-to-point form, the default mode is point-to-point. The default cost is 1.

Context

ospf area statement

Examples

Example 1

```
ospf yes {  
    area 2.2.2.2 {  
        interface fxp3 cost 1 {  
            retransmitinterval 30;  
            transitdelay 2;  
            priority 62;  
            hellointerval 10;  
            routerdeadinterval 30;  
            auth simple "simple";  
        };  
    };  
};
```

Example 2

```
ospf yes {  
    area 2.2.2.2 {  
        interface fxp0 nonbroadcast cost 1 {  
            routers {  
                192.168.0.1 ;  
                192.168.1.1 ;  
            };  
        };  
    };  
};
```

Example 3

```
ospf yes {  
    backbone {  
        interface fxp0 cost 1 { passive; };  
    };  
};
```

See Also

- area on page 98
- retransmitinterval on page 130
- transitdelay on page 148
- priority on page 128
- hellointerval on page 107
- routerdeadinterval on page 133
- pollinterval on page 126
- auth on page 99
- routers on page 135
- strict-routers on page 137

networks

Name

networks - describes the networks comprising an area on an Area Border Router (ABR)

Syntax

```
networks {  
    network mask mask [ restrict ];  
    network masklen number [ restrict ];  
    host nethost [ restrict ] ;  
    [...]  
};
```

Parameters

network - the network prefix to summarize

mask - mask of the network prefix

number - mask length of the network prefix

nethost - host route to be summarized

Description

The **networks** list describes the scope of an area on an ABR. Intra-area LSAs that fall within the specified ranges are not advertised into other areas as inter-area routes. Instead, the specified ranges are advertised as summary network LSAs. If **restrict** is specified, the summary network LSAs and all LSAs within the range are not advertised. Intra-area LSAs that do not fall into any range are also advertised as summary network LSAs. On well-designed networks, **networks** reduces the amount of routing information propagated between areas. The entries in this list are either networks, subnetwork/mask pairs, or subnetwork/masklen pairs. See Chapter 26, "Route Filtering" on page 129 of *Configuring GateD* for more detail about specifying ranges. Specifying **networks** on a non-ABR will have no effect.

Default

none

Context

ospf area statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        networks {  
            10.0.0.0 mask 255.0.0.0;
```

```
        192.168.0.0 mask 255.255.0.0 restrict;  
        172.14.27 masklen 24;  
    };  
    interface fxp0 cost 1;  
};  
};
```

See Also

`nssanetworks` on page 121

`area` on page 98

nssa

Name

nssa - configures the area as a Not-So-Stubby-Area (NSSA) according to RFC 1587

Syntax

```
nssa [ cost defaultcost type 1 | 2 ] ;
```

Parameters

defaultcost - specifies that a type-7 default should be originated into the NSSA area with the given cost

type 1 | 2 - the type of metric to be used in the default NSSA LSA

Description

nssa configures the area as an NSSA according to RFC 1587. If the router is an Area Border Router (ABR) and has the highest Router-ID of all the ABRs in the area, it will translate a type-7 LSA to type-5 LSA. The translation is affected by the **nssanetworks** clause, which is similar in operation to the **networks** clause. **nssa** and **stub** are mutually exclusive.

When **cost** is specified, the type of NSSA metric must be given with **type**. This sets the type of NSSA metric originated in the default NSSA LSA.

Default

The default type of area is non-stub non-NSSA. If **nssa** is specified, the default is no advertisement of a type-7 default LSA.

Context

ospf area statement

Examples

```
ospf yes {  
    area 1.1.1.1 {  
        nssa cost 1 type 1 ;  
        interface 10.1.1.1 cost 1 ;  
    };  
};
```

See Also

area on page 98

stub on page 138

nssanetworks on page 121

nssa-cost

Name

nssa-cost - specifies the default cost when importing non-OSPF routes into OSPF NSSA

Syntax

```
nssa-cost defnssacost ;
```

Parameters

defnssacost - default cost of routes exported into a Not So Stubby Area (NSSA)

Description

nssa-cost is used when exporting a non-OSPF route from the GateD routing table into OSPF as an NSSA route. **nssa-cost** can be explicitly overridden in export policy.

Default

```
cost 1 ;
```

Context

```
ospf defaults statement
```

Examples

```
ospf yes {  
    defaults {  
        nssa-cost 1;  
    };  
    area 1.2.3.4 {  
        interface fxp0;  
    };  
};
```

See Also

defaults on page 103

cost on page 102

nssa-inherit-metric

Name

nssa-inherit-metric - configures an OSPF NSSA route to inherit the metric of the external route when no metric is specified on the export policy

Syntax

```
inherit-metric ;
```

Parameters

none

Description

nssa-inherit-metric allows an OSPF NSSA route to inherit the metric of the external route when no metric is specified on the export policy. **nssa-inherit-metric** maintains compatibility with all the current export functions. A metric specified on the export policy will take precedence. The **cost** specified in the **default** statement (*defnssacost*) will be used if **nssa-inherit-metric** is not specified.

Default

The default is to not inherit the external metric.

Context

ospf defaults statement

Examples

```
ospf yes {  
    defaults {  
        nssa-inherit-metric;  
    };  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

defaults on page 103

cost on page 117

inherit-metric on page 109

nssa-preference

Name

nssa-preference - specifies how preferred OSPF NSSA routes will be, compared to other protocols, when selecting active routes

Syntax

```
nssa-preference defnssapref ;
```

Parameters

defnssapref - default NSSA preference

Description

preference specifies how active routes that are learned from the OSPF NSSA (compared to other protocols) 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.

Default

```
nssa-preference 150 ;
```

Context

ospf defaults statement

Examples

```
ospf yes {  
    defaults {  
        nssa-preference 160;  
    };  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

defaults on page 103

preference on page 127

"Preferences and Route Selection" on page 11 of *Configuring GateD*

nssa-type

Name

nssa-type - changes the default type of routes exported from the GateD routing table into OSPF NSSA

Syntax

```
nssa-type 1 | 2 ;
```

Parameters

1 | 2 - sets the default metric type to 1 or 2

Description

Routes exported from the GateD routing table into OSPF default to becoming type 1 NSSAs. This default can be explicitly changed here and overridden in export policy.

Default

```
nssa-type 1
```

Context

```
ospf defaults statement
```

Examples

```
ospf yes {  
    defaults {  
        nssa-type 2;  
    };  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

defaults on page 103

type on page 150

nssanetworks

Name

nssanetworks - lists the net ranges that should be translated into type-5 LSAs from NSSA type-7 LSAs

Syntax

```
nssanetworks {
    network mask stubmask [ restrict ] ;
    network masklen number [ restrict ] ;
    host stubhost [ restrict ] ;
};
```

Parameters

network - specifies the network range to be aggregated into type-5

number - specifies the subnetwork mask length

restrict - specifies that this range should not be summarized into type-5

stubmask - specifies the subnetwork mask

stubhost - host route to be summarized

Description

nssanetworks lists the net ranges that should be translated into type-5 LSAs from NSSA type-7 LSAs. The default behavior is to translate type-7 LSAs that do not fall within a configured net range. This clause is valid only in an NSSA. It will be ignored when set in a non-NSSA.

Default

There are no default ranges. You must specify a range if **nssanetworks** is used.

The default cost is specified by the **nssa-cost** clause in the **defaults** clause.

Context

ospf area statement

Examples

```
ospf yes {
    area 1.2.3.4 {
        nssa;
        nssanetworks {
            10.0.0.0 mask 255.0.0.0 restrict;
            192.168.0.0 mask 255.255.0.0;
        };
        interface fxp0 cost 1;
    };
};
```

```
};  
};
```

See Also

area on page 98

nssa on page 138

nssa-cost on page 117

opaque-capability

Name

`opaque-capability` - configures support for RFC 2370 Opaque LSAs

Syntax

```
opaque-capability on | off ;
```

Parameters

`on` | `off` - enables or disables the option

Description

`opaque-capability` configures support for RFC 2370, "Opaque LSAs". An internal API is provided for viewing and originating/flushing these types of LSAs. Because this can unnecessarily increase the size of the router's Link-State Database, and it does not affect normal protocol operation, the default is `off`. Routers that do not support Opaque LSAs should continue to interoperate with those that do support them.

Default

```
opaque-capability off
```

Context

`ospf` statement

Examples

```
ospf yes {  
    opaque-capability on;  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

`area` on page 98

`interface` on page 110

`ospf` on page 124

ospf

Name

ospf - enables or disables OSPF

Syntax

```
ospf on | off { ospf_parameters }
```

Parameters

ospf_parameters - all parameters in this section

Description

The **ospf** statement enables or disables OSPF. By default OSPF is disabled.
ospf_parameters includes all the parameters in this section.

Default

```
ospf off ;
```

Context

global

Examples

```
ospf yes ;
```

See Also

area on page 98

interface on page 110

passive

Name

`passive` - disables reception and transmission on an interface

Syntax

```
passive ;
```

Parameters

none

Description

`passive` causes GateD to not send or receive packets on this interface. For example, `passive` is used when this is the only router on the network. `passive` has the effect of originating a stub link to this interface into the domain.

Note: OSPF `passive` is not used to learn other routers' announcements, which is the way RIP `passive` is used. To learn about routes, if your host is connected to a single network on which there are multiple routers, use Router Discovery combined with ICMP redirects to learn a default route and the best route. If your host is connected directly to multiple networks, this method might not produce the best routes.

Default

The default is non-passive.

Context

`ospf area interface` statement

`ospf virtuallink` statement

Examples

```
ospf yes {
    backbone {
        interface fxp0 cost 1 { passive; };
    };
};
```

See Also

`interface` on page 110

`hellointerval` on page 107

pollinterval

Name

pollinterval - the length of time, in seconds, between OSPF packets that the router sends before adjacency is established with a neighbor

Syntax

```
pollinterval time ;
```

Parameters

time - time, in seconds, between hello packets

Description

pollinterval is the length of time, in seconds, between OSPF packets that the router sends before adjacency is established with a neighbor. **pollinterval** can be specified at the global level and can be overridden at the area and interface levels. This can be used to reduce network overhead in cases where a router may or may not have a neighbor on a given interface at the expense of initial convergence time.

Default

```
pollinterval 120 ;
```

Context

ospf statement

area statement

interface statement

virtuallink statement

Examples

```
ospf yes {  
    backbone {  
        interface fxp0 cost 1 {  
            pollinterval 120;  
        };  
    };  
};
```

See Also

interface on page 110

hellointerval on page 107

preference

Name

preference - specifies how preferred OSPF ASE routes will be, compared to other protocols, when selecting active routes

Syntax

```
preference defasepref ;
```

Parameters

defasepref - default ASE preference

Description

preference specifies how active routes that are learned from the OSPF ASE (compared to other protocols) 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.

Default

```
preference 150 ;
```

Context

ospf defaults statement

Examples

```
ospf yes {  
    defaults {  
        preference 160;  
    };  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

defaults on page 103

nssa-preference on page 119

"Preferences and Route Selection" on page 11 of *Configuring GateD*

priority

Name

priority - a number between 0 and 255 that specifies the priority for becoming the designated router (DR)

Syntax

```
priority level ;
```

Parameters

level - priority level for becoming DR

Description

priority is a number between 0 and 255 that specifies the priority for becoming the DR. When more than one router attached to a network attempts to become the DR, the one with the highest priority wins. If the competing routers have the same priority, the one with the highest router ID becomes the DR. The router coming in second in the election becomes the backup DR. A router with a router priority set to 0 is ineligible to become the DR. **priority** can be specified at the global level and overridden at the area and interface levels.

Note: **priority** applies only to broadcast or NBMA media.

Default

```
priority 0 ;
```

Context

ospf statement

area statement

interface statement

Examples

```
ospf yes {
    area 1.2.3.4 {
        priority 1;
        interface fxp0 cost 1;
        interface fxp1 cost 2 { priority 2; };
    };
    backbone {
        interface fxp2 cost 1;
    };
};
```

See Also

interface on page 110

area on page 98

retransmitinterval

Name

retransmitinterval - sets the default for the number of seconds between link state advertisement retransmissions for adjacencies

Syntax

```
retransmitinterval time ;
```

Parameters

time - time, in seconds, for retransmission interval

Description

retransmitinterval sets the default for the number of seconds between link state advertisement retransmissions for adjacencies. If a Link State Protocol (LSP) is not acknowledged within **retransmitinterval** seconds, it is resent. This setting is another convergence/network traffic trade-off. **retransmitinterval** may be specified at the global level and overridden at the area and interface levels.

Default

```
retransmitinterval 5 ;
```

Context

ospf statement

ospf area statement

ospf area interface statement

ospf area virtuallink statement

Examples

```
ospf yes {  
    backbone {  
        retransmitinterval 10;  
        interface fxp0 cost 1;  
    };  
};
```

See Also

interface on page 110

area on page 98

virtuallink on page 151

rfc1583compatibility

Name

`rfc1583compatibility` - enables RFC 1583 compatibility mode for the SPF calculation

Syntax

```
rfc1583compatibility on | off ;
```

Parameters

`on` | `off` - toggles whether or not the RFC 1583 preference rules are used

Description

Set `rfc1583compatibility` to `off` if all the routers using an OSPF implementation in your domain are based on RFC 2328 or later. This option should be set the same way on all routers in the domain. If any of the routers do not have this option, you should always choose `on`. When disabled, the preference rules for best route election are changed to eliminate certain kinds of possible routing loops.

Default

```
rfc1583compatibility on ;
```

Context

`ospf` statement

Examples

```
ospf on {  
    rfc1583compatibility on;  
    area 1.2.3.4 {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

`ospf` on page 124

ribs

Name

ribs - specifies the Routing Information Base (RIB) in which OSPF internal routes are installed

Syntax

```
ribs unicast [ multicast ] ;
```

Parameters

unicast - specifies that routes should be installed in the unicast RIB

multicast - specifies that routes should be installed in the multicast RIB

Description

ribs specifies the RIB in which OSPF internal routes are installed. The unicast RIB is required and is the default. In code bases that support extended RIBs, OSPF routes can be installed in the multicast RIB. **ribs** has no effect on self-originated ASE or NSSA routes, because they are exported from another protocol.

Default

```
ribs unicast
```

Context

```
ospf defaults statement
```

Examples

```
ospf yes {
    defaults {
        ribs unicast multicast;
    };
    backbone {
        interface fxp0 cost 1;
    };
};
```

See Also

defaults on page 103

routerdeadinterval

Name

routerdeadinterval - configures the number of seconds that may elapse without receiving a neighbor's hello packets before the neighbor will be declared down

Syntax

```
routerdeadinterval time ;
```

Parameters

time - time, in seconds, for the interval

Description

routerdeadinterval is the number of seconds that may elapse without receiving a router's hello packets before the router's neighbors will declare it down. **routerdeadinterval** can be specified at the global level and overridden at the area and interface levels. A general rule is for **routerdeadinterval** to equal three times the HELLO interval. Do not set **routerdeadinterval** to be less than **hellointerval**, or convergence will not occur.

Default

```
routerdeadinterval 40 ;
```

Context

```
ospf statement
ospf area statement
ospf area interface statement
ospf area virtuallink statement
```

Examples

```
ospf yes {
    backbone {
        interface fxp0 cost 1 {
            routerdeadinterval 20;
        };
    };
};
```

See Also

interface on page 110
hellointerval on page 107

area on page 98

virtuallink on page 151

routers

Name

routers - specifies one or more neighbors and an indication of their eligibility to become a designated router

Syntax

```
routers {
    gateway [eligible];
    [...]
};
```

Parameters

gateway - the IP address of another router on the network

eligible - specifies whether the *gateway* router is eligible to become a designated router (DR) (nonbroadcast mode only).

Description

By definition, it is not possible to send broadcast or multicast packets to discover OSPF neighbors on a nonbroadcast medium, so all neighbors must be configured. The gateway list includes one or more neighbors and an indication of their eligibility to become a DR.

Default

There are no default addresses. The default eligibility is not eligible.

Context

ospf area interface nonbroadcast statement

ospf area interface point-to-multipoint statement

Examples

```
ospf yes {
    backbone {
        interface fxp0 nonbroadcast cost 1 {
            routers {
                10.1.1.1 eligible;
            };
        };
    };
};
```

See Also

`interface` on page 110

`priority` on page 128

strict-routers

Name

strict-routers - configures GateD to ignore packets from routers not specified in the **routers** statement

Syntax

```
strict-routers on | off ;
```

Parameters

on | **off** - enables/disables the option

Description

If **on** (the default), GateD ignores packets from routers not specified in the **routers** statement.

Default

```
strict-routers on
```

Context

ospf area interface nonbroadcast statement

ospf area interface point-to-multipoint statement

Examples

```
ospf yes {
    backbone {
        interface fxp0 point-to-multipoint cost 1 {
            routers {
                10.1.1.1;
            };
            strict-routers on;
        };
    };
};
```

See Also

interface on page 110

routers on page 135

stub

Name

stub - configures an area as a stub area

Syntax

```
stub [ cost stub_default_cost ] ;
```

Parameters

stub_default_cost - cost of type-3 default summary to advertise into the area

Description

A **stub** area is one in which there are no ASE or NSSA routes. Each router in the area must specify that the area is a stub, or adjacencies will not form. If a **cost** is specified, **cost** is used to inject a default route into the area with the specified cost originating from this router. **cost** should be specified only on an Area Border Router (ABR). It is possible to use **stub** on multiple ABRs and give them different **costs**. **stub** and **nssa** are mutually exclusive.

Default

The default type of area is non-stub and non-NSSA. The default for a stub area is to not advertise a type-3 summary.

Context

ospf area statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        stub cost 1;  
        interface fxp0 cost 1;  
    };  
};
```

See Also

area on page 98

nssa on page 116

stubhosts

Name

stubhosts - specifies directly attached hosts that should be advertised as reachable from this router and the costs with which they should be advertised

Syntax

```
stubhosts {  
    host cost cost;  
};
```

Parameters

host - IP address of host to advertise in router LSA

cost - cost of link to be advertised

Description

The **stubhosts** list specifies directly attached hosts that should be advertised as reachable from the router and the costs with which they should be advertised. Point-to-point interfaces on which it is not desirable to run OSPF should be specified here. It is also useful to assign an additional address to the loopback interface (one not on the 127 network) and advertise it as a stubhost. If this address is the same one used as the router ID, it enables routing to OSPF routers by router ID, instead of by interface address. Routing by router ID is more reliable than routing to one of the router's interface addresses, which may not always be reachable.

Default

none

Context

ospf area statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        stubhosts {  
            10.1.1.1 cost 1;  
        };  
        interface fxp0 cost 1;  
    };  
};
```

See Also

`stubnetworks` on page 141

`stub` on page 138

stubnetworks

Name

stubnetworks - specifies directly attached networks that should be advertised as reachable from this router and the costs with which they should be advertised

Syntax

```
stubnetworks {  
    network mask stubmask cost cost ;  
    network masklen number cost cost ;  
    host stubhost cost cost ;  
};
```

Parameters

network - network to be advertised

stubmask - netmask of the network to be advertised

number - mask length of network to be advertised

cost - cost of the network

stubhost - host route to be summarized

Description

The **stubnetworks** list specifies directly attached networks that should be advertised as reachable from this router and the costs with which they should be advertised. Interfaces on which it is not desirable to run OSPF should be specified here. No checking is currently done on whether the specified network is actually reachable from this router, so care should be taken.

Default

none

Context

ospf area statement

Examples

```
ospf yes {  
    backbone {  
        stubnetworks {  
            10 mask 255.0.0.0 cost 1;  
            192.168 masklen 16 cost 5;  
        };  
        interface fxp0 cost 1;  
    };  
};
```

};

See Also

`stubhosts` on page 139

`passive` on page 125

summaryfilters

Name

summaryfilters - contains route filters that specify which summary LSAs to filter from a non-transit area

Syntax

```
summaryfilters {  
    route_filter  
    [...]  
};
```

Parameters

route_filter - See the section on route filters for syntax.

Description

The **summaryfilters** statement contains route filters that specify which summary LSAs to filter from the a non-transit (typically **stub** or **nssa**) area. That is, if a summary would normally be injected into the area, it is compared against the summary-filters list, and if a match is found, the announcement of the summary LSAs into the stub area will be suppressed. For normal operation, summary-filters should only be used in **stub** or **nssa** areas that have a default route being generated (see **stub** or **nssa**). In this usage, you can filter all summary LSAs (not including the generated default) to further reduce the amount of routing information present in the stub area's routers. Use of **summaryfilters** in non-stub non-NSSA areas is not recommended because it can break routing.

Default

By default all summary is passed into the area.

Context

ospf area statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        stub cost 1;  
        summary-filters {  
            10.0.0.0 mask 255.0.0.0;  
        };  
        interface fxp0 cost 1;  
    };  
};
```

See Also

stub on page 138

nssa on page 116

area on page 98

“Chapter 28 Route Filtering” on page 129 of “Configuring GateD”

tag

Name

tag - used to propagate data from an exterior gateway protocol (such as BGP) through OSPF

Syntax

```
tag [ as ] tagvalue ;
```

Parameters

as - specifies that *tagvalue* is a BGP AS number

tagvalue - a 32-bit value for the tag

Description

OSPF ASE routes have a 32-bit tag field that is not used by the OSPF protocol, but can be used when exporting to protocols other than OSPF. When OSPF is interacting with BGP, the **tag** field can be used to propagate AS path information, in which case the **as** keyword is specified, and the tag is limited to 12 bits of information. If not specified, the tag is set to 0.

Default

```
tag [ as ] 0 ;
```

Context

ospf defaults statement

Examples

```
ospf yes {
    defaults {
        tag 2112;
    };
    area 1.2.3.4 {
        interface fxp0 cost 1;
    };
};
```

See Also

defaults on page 103

traceoptions

Name

traceoptions - specifies the tracing options for OSPF

Syntax

```
traceoptions trace_options_ospf ;
```

Parameters

trace_options_ospf - tracing options for the OSPF protocol

Description

traceoptions specifies the tracing options for OSPF.

The following are OSPF-specific traceoptions:

lsabuild - Trace the creation of link-state advertisements.

lsatransmit (or **lsatx**) - Trace the link-state packets transmitted.

lsareceive (or **lsarx**) - Trace the link-state packet received.

spf - Trace the Shortest Path First (SPF) calculations.

debug - Trace OSPF at the debugging level of detail.

Packet tracing options (which can be modified with **detail**, **send**, and **recv**) include:

hello - Trace OSPF hello packets, which are used to determine neighbor reachability.

dd - Trace OSPF Database Description (DD) packets, which are used in synchronizing OSPF databases.

request - Trace OSPF link-state request packets, which are used in synchronizing OSPF databases.

lsu - Trace OSPF link-state update packets, which are used in synchronizing OSPF databases.

ack - Trace OSPF link-state ack packets, which are used in synchronizing OSPF databases.

Default

The default trace options are inherited from the global trace options.

Context

ospf statement

Examples

```
ospf yes {  
    traceoptions "/tmp/log" all;  
    area 1.2.3.4 {  
        interface fxp0 cost 1;  
    };  
};
```

```
};
```

See Also

“Trace Statements” on page 15 of *Configuring GateD*

transitdelay

Name

transitdelay - sets the estimated number of seconds required to transmit a link state update

Syntax

```
transitdelay time ;
```

Parameters

time - time, in seconds, for the transit delay

Description

transitdelay sets the estimated number of seconds required to transmit a link state update. **transitdelay** takes into account transmission and propagation delays and must be greater than 0. **transitdelay** can be specified at the global level and overridden at the area and interface levels.

Default

```
transitdelay 1 ;
```

Context

ospf statement

ospf area statement

ospf area interface statement

ospf area virtuallink statement

Examples

```
ospf yes {  
    area 1.2.3.4 {  
        transitdelay 5;  
        interface fxp0 cost 1;  
        interface fxp1 cost 2 {  
            transitdelay 10;  
        };  
        interface fxp2 cost 3;  
    };  
};
```

See Also

interface on page 110

area on page 98

virtuallink on page 151

type

Name

type - changes the default type of routes exported from the GateD routing table into OSPF ASE

Syntax

```
type 1 | 2 ;
```

Parameters

1 | 2 - sets the default metric type to 1 or 2

Description

Routes exported from the GateD routing table into OSPF default to becoming type 1 ASEs. This default can be explicitly changed here and overridden in export policy.

Default

```
type 1
```

Context

```
ospf defaults statement
```

Examples

```
ospf yes {  
    defaults {  
        type 2;  
    };  
    backbone {  
        interface fxp0 cost 1;  
    };  
};
```

See Also

defaults on page 103

nssa-type on page 120

virtuallink

Name

virtuallink - configures a virtual link on the backbone

Syntax

```
virtuallink neighborid router_id transitarea area
```

Parameters

router_id - specifies the router-ID of the other end of this link

area - specifies the transit area to be used for the link

Description

Virtual links are used to establish or increase connectivity of the backbone area. The **neighborid** is the **router_id** of the other end of the virtual link. The area specified in the **transitarea** must also be configured on the system. All standard interface parameters defined by the **interface** clause can be specified on a virtual link.

Note: **virtuallink** is available on the backbone only.

Default

none

Context

ospf area statement

Examples

```
ospf yes {  
    backbone {  
        interface fxp0 cost 1;  
        virtuallink neighborid 2.1.1.2 transitarea 1.2.3.4;  
    };  
    area 1.2.3.4 {  
        interface fxp1 cost 2;  
    };  
};
```

See Also

interface on page 110

Chapter 9

Intermediate System to Intermediate System (IS-IS)

area

Name

area - configures the IS-IS area of the router

Syntax

area *D.D.D.D* ;

area *HH.HHHH.HHHH.HHHH.HHHH* ;

Parameters

D - a decimal integer between 0 and 255. The **area** *D.D.D.D* form of the command is supplied to be interchangeable with the **area** command in OSPF. (See "OSPF" on page 45 in *Configuring GateD*.) This form of **area** takes as its argument a dotted-quad (IPv4-like) address. This form will set the area to be 4 octets as given by the four decimal numbers in the dotted-quad argument.

H - a hexadecimal digit between 0 and F. The more general form of **area** uses hexadecimal digits and is variable in length. The area will be as long as the given argument dictates. Each pair of hexadecimal digits is a single octet, and thus an even number of hex-digits must be given.

Description

area sets the area of the router. In OSI, each router (IS) has a network entity title (NET) assigned to it. The NET uniquely identifies the IS throughout the routing domain and determines which area the router is in.

A NET is divided into the area portion, then the system ID, and completed by a single 0 octet (byte). The system ID is always (by convention) 6 octets. Thus, the area portion is variable and can be from 1 to 13 octets.

Some routers require that the user specify the area and system ID by setting the NET directly. This mechanism is for the most part outdated because most organizations do not actually run an OSI routing domain; it has been simplified to resemble other current routing protocols, such as OSPF.

Note: This option can appear up to three times.

Default

```
area 00.0000.0000.0000 ;
```

Context

`isis` statement

Examples

Example 1

The following example configures the router to be in area 0.0.0.3 (i.e., 3 octets of 0 followed by 1 of 3).

```
isis on {  
    area 0.0.0.3;  
};
```

Example 2

This example configures the router's area using the hexadecimal form; its length is 5 octets and the final octet has the value 31.

```
isis on {  
    area 00.0000.001F;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` on page 185

`systemid` on page 222

area auth

Name

area auth - configures the area password(s)

Syntax

```

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] ; ]
    } ; ]
} ;

```

Parameters

key - a password of length 1 to 255, given inside double quotes

Description

IS-IS authentication is divided up into three sections: hello authentication, level 1 authentication and level 2 authentication.

area auth configures the level 1 authentication. All non-hello level 1 packets are authenticated using the **area auth key**. This includes level 1 link state packets (LSPs), level 1 partial sequence number (PSN) packets, and level 1 complete sequence numbers (CSN) packets.

All level 1 routers in the area should be configured with the same key. If multiple keys are specified, the first key in the list is used to authenticate transmitted packets and all the keys in the list are used to authenticate received packets. Many older (and non-standard) implementations do not actually authenticate the PSN and CSN packets, thus another command (**require-snp-auth** - see page 203) is provided to enable or disable SNP authentication. For interoperability reasons, SNP authentication is off by default.

The MD5 time ranges may be used to roll over one key to another. The time ranges are a convenience that allow the router to automatically roll over the key. A sufficient window of time should be used in which routers are accepting both keys to allow a smooth transition. The expiration of a generate time does not cause level 1 LSP to be regenerated; therefore, a sufficient window of time should be allocated to allow for normal refreshing of the LSP, which would cause the new authentication to be used.

Default

no authentication

Context

isis statement

Examples

The following example configures the area password to be "my-area-password".

```
isis on {  
    area auth simple "my-area-password" ;  
};
```

See Also

auth on page 157

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

domain auth on page 166

isis on page 185

require-snp-auth on page 203

auth

Name

auth - configures the interface password(s)

Syntax

```
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 ) ] ;
}
```

Parameters

key - a password of length 1 to 255, given inside double quotes

Description

IS-IS authentication is divided into three sections: hello authentication, level 1 authentication, and level 2 authentication.

auth configures the hello authentication. All hello packets are authenticated using the **auth key**. The level subcommands allow separate level 1 and level 2 hello authentication keys to be specified.

All routers at a given level on a link should be configured with the same key. If multiple keys are specified, the first key in the list is used to authenticate transmitted packets and all the keys in the list are used to authenticate received packets.

The MD5 time ranges may be used to roll-over one key to another. The time ranges are a convenience that allow the router to automatically roll over the key. A sufficient window of time should be used where routers are accepting both keys to allow a smooth transition.

Default

no authentication

Context

isis interface statement

Examples

Example 1

The following example configures the interface eth0's password to "my-intf-password" for level 1 and 2.

```
isis on {  
    interface eth0 {  
        auth simple "my-intf-password" ;  
    };  
};
```

Example 2

This example configures eth0's level 1 interface password to be "my-level-1-password" and the level 2 interface password to be "my-level-2-password".

```
isis on {  
    interface eth0 {  
        auth simple "my-level-1-password" level 1;  
        auth simple "my-level-2-password" level 2;  
    };  
};
```

Example 3

This example enables md5 authentication on interface "eth0" for level 1 and 2.

```
isis on {  
    interface eth0 {  
        auth md5 key "my-key" level 1 and 2;  
    };  
};
```

Example 4

This example enables key rollover on interface "eth0".

```
isis on {  
    interface eth0 {  
        auth md5 key "key-1" {  
            start-generate 2001/10/1 12:00 ;  
            stop-generate 2001/10/2 12:00 ;  
            start-accept 2001/10/1 12:00 ;  
            stop-accept 2001/10/3 12:00 ;  
        };  
        auth md5 key "key-2" {
```

```
        start-generate 2001/10/1 12:00 ;
        stop-generate 2002/10/2 12:00 ;
        start-accept 2001/10/1 12:00 ;
        stop-accept 2002/10/3 12:00 ;
    };
};
};
```

See Also

area auth on page 155

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

domain auth on page 166

interface on page 184

isis on page 185

config-time

Name

`config-time` - used to set the ISO IS config time in the operating system kernel

Syntax

```
config-time seconds;
```

Parameters

seconds - the value of the config time

Description

This value is used to set the ISO IS config time in the operating system kernel. This requires ISO socket support in the operating system.

Default

```
config-time 60;
```

Context

`isis` statement

Examples

```
isis on {
    config-time 20;
    interface ex0 cost 1;
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` on page 185

csn-interval

Name

csn-interval - configures the Complete Sequence Number (CSN) interval for the interface

Syntax

```
csn-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

intervalnum - a number of seconds between 1 and 100

Description

The Designated Intermediate System (DIS) on a broadcast link periodically multicasts CSN packets to the link. The CSN packets summarize the current link state packet (LSP) database as it appears on the DIS. This allows the other routers on the link to request updates or to know that they should send updates. This method keeps all the routers on the link synchronized (for example, having the same database).

Every *intervalnum* seconds, the DIS will multicast as many CSN packets as are required to completely describe its LSP database. Setting *intervalnum* lower can decrease convergence time at the expense of more link usage. Setting *intervalnum* higher will decrease link usage at the expense of convergence time. The level subcommands allow separate *intervalnum* intervals for each level at which the interface operates. If no level is specified, **level 1 and 2** is assumed.

This command is typically ignored on point-to-point links because CSNPs are transmitted only upon establishment of the adjacency. In order for CSN to be transmitted periodically on point-to-point links, according to the **csn-interval** parameter, the **periodic-csn** command must be used.

Default

```
csn-interval 10 level 1 and 2;
```

Context

isis interface statement

Examples

Example 1

The following example sets the CSN interval to 20 seconds for all levels.

```
isis on {
    interface eth0 {
        csn-interval 20;
    }
}
```

```
};
```

Example 2

This example sets the level 1 CSN interval to 15 seconds, and sets the level 2 CSN interval to 20 seconds.

```
isis on {  
    interface eth0 {  
        csn-interval 15 level 1;  
        csn-interval 20 level 2;  
    }  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

interface on page 184

isis on page 185

priority on page 200

periodic-csn on page 202

disable

Name

disable - disables IS-IS on an interface

Syntax

```
disable ;
```

Parameters

none

Description

If **disable** is specified in the **interface** statement, the interface will not be considered a part of the IS-IS routing domain.

Default

not applicable

Context

isis interface statement

Examples

The following example disables interface eth0.

```
isis on {  
    interface eth0 {  
        disable;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

enable on page 169

interface on page 184

isis on page 185

dis-hello-interval

Name

dis-hello-interval - configures the hello interval for a designated intermediate system (DIS) interface

Syntax

```
dis-hello-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

intervalnum - a number of seconds between 1 and 100

Description

When a router is elected DIS for a link, the router switches to a different hello interval than in the non-DIS case. **dis-hello-interval** allows you to set the new interval length to use when becoming DIS. The level subcommand allows separate intervals for each level.

The **dis-hello-interval** is used to calculate the hold time announced in hello packets as follows:

hold time = **dis-hello-interval** * **hello-multiplier**

Each hello packet contains a hold time. The hold time informs the receiving routers how long to wait without seeing another hello from the sending router before considering the sending router down.

Default

```
dis-hello-interval 3 level 1 and 2 ;
```

Context

isis interface statement

Examples

Example 1

The following example sets the DIS hello interval on interface eth0 to 5 seconds.

```
isis on {  
    interface eth0 {  
        dis-hello-interval 5;  
    }  
};
```

Example 2

This example sets per-level DIS hello intervals for interface eth0. For level 1, the interval is set to 4; for level 2, the interval is set to 6.

```
isis on {  
    interface eth0 {  
        dis-interval 4 level 1;  
        dis-hello-interval 6 level 2;  
    }  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

hello-interval statement on page 176

hello-multiplier statement on page 178

interface statement on page 184

isis statement on page 185

priority on page 200

domain auth

Name

`domain auth` - configures the domain password(s)

Syntax

```
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] ; ]
    } ; ]
} ;
```

Parameters

key - a password of length 1 to 255, given inside double quotes

Description

IS-IS authentication is divided into three sections: hello authentication, level 1 authentication, and level 2 authentication.

`domain auth` configures the level 2 authentication. All non-hello level 2 packets are authenticated using the `domainauth key`. This includes level 2 LSPs, level 2 partial sequence numbers (PSN) packets, and level 2 complete sequence numbers (CSN) packets. All level 2 routers in the routing domain should be configured with the same key. If multiple keys are specified, the first key in the list is used to authenticate transmitted packets, and all the keys in the list are used to authenticate received packets. Many older (and non-standard) implementations do not actually authenticate the PSN and CSN packets; thus, another command (`require-snp-auth` - see page 203) is provided to enable or disable SNP authentication. For interoperability reasons, SNP authentication is off by default.

Default

no authentication

Context

`isis` statement

Examples

The following example configures the domain password to be "my-domain-password".

```
isis on {  
    area auth simple "my-domain-password" ;  
};
```

See Also

`area auth` on page 155

`auth` on page 155

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` on page 185

`require-snp-auth` on page 203

domain-wide

Name

domain-wide - enables or disables domain-wide extension

Syntax

```
domain-wide ( on | off ) ;
```

Parameters

none

Description

The **domain-wide** statement enables support for redistributing routes from level 2 down into level 1. This method can be used to achieve more optimal routing. Normally (with **domain-wide off**), level 1 routers send packets destined for networks not within their area to the closest level 2 router. If multiple level 2 routers are active for the area, the closest level 2 router may not actually be the closest in overall path cost.

A level 2 router can filter some of the prefixes advertised into the level 1 area through the use of the **summary-filter** statement (see page 216).

All level 2 routers must be configured the same way, with domain-wide routing either **on** or **off**. If all level 2 routers are not configured the same way, forwarding loops may arise.

This functionality is described in RFC 2966.

Default

```
domain-wide off;
```

Context

isis statement

Examples

The following example configures the router to advertise level 2 routes into its level 1 area.

```
isis on {  
    domain-wide on;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

isis on page 185

summary-filter on page 216

enable

Name

enable - enables IS-IS on an interface

Syntax

enable ;

Parameters

none

Description

If **enable** is specified in the **interface** statement, the interface will be considered a part of the IS-IS routing domain. **enable** is present for completeness. (**enable** is the opposite of **disable**.)

Default

enable ;

Context

isis interface statement

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

disable statement on page 163

interface statement on page 184

isis statement on page 185

es-config-time

Name

es-config-time - used to set the ISO ES config time in the operating system kernel

Syntax

```
es-config-time seconds;
```

Parameters

seconds - the value of the ES config time

Description

This value is used to set the ISO ES config time in the operating system kernel. This requires ISO socket support in the operating system.

Default

```
es-config-time 60;
```

Context

isis statement

Examples

```
isis on {  
    es-config-time 60;  
    interface ex0 cost 1;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

export-defaults

Name

`export-defaults`

Syntax

```
export-defaults metric-type ( internal | external );
export-defaults metric ( metricnum | inherit );
export-defaults level ( 1 | 2 );
```

Parameters

metric-type [**internal** | **external**] - default metric type for exported reachability

metric *metricnum* - default metric for exported reachability, an integer between 1 and *max_metric*

metric inherit - default metric for exported reachability

level [**1** | **2**] - default level into which to export exported reachability

Description

This command sets the defaults for exportation of exported reachability. The values configured in the export clause may override what is given here.

export-defaults metric-type configures the default metric type that routes from other protocols receive when they are exported into IS-IS. The default metric type is used only when the user does not specify the metric type in the **export** statement. IS-IS has two distinct metric types, **internal** and **external**. Routes with internal metrics are preferred over routes with external metrics at the same level. Interface metrics advertised by the IS are considered **internal**. If the exported route will have a metric compatible with the interface metrics, it should be exported with metric type **internal**. If the metric is not compatible with interface metrics, it should be exported with metric type **external**.

In any case, routes originating in level 1 are always preferred over level 2, and routes originating in level 2 are always preferred over routes being summarized back into level 1. Note that the same route originated in IS-IS (i.e., an interface route) and exported into IS-IS with internal metric will be directly compared. The exported version could be preferred to the originated version if the cost is smaller.

export-defaults metric configures the default metric that routes from other protocols receive when they are exported into IS-IS. The default metric is used only when the user does not specify the metric in the **export** statement. The **inherit** form of the statement causes the source protocol's **metric** to be used. It is assumed that the metrics are compatible in meaning. If the source protocol's **metric** exceeds *max_metric*, the exported metric will be set to *max_metric*, where *max_metric* is 63 if **rfc1195-metrics** is enabled; otherwise, it is $2^{24}-1$ (or 4,261,412,864).

The **export-defaults level** statement configures the default level into which routes from other protocols are injected. The default level is used only when the user does not specify the level on the **export** statement. Exporting routes into level 1 is not strictly standard; however, most implementations of IS-IS support it.

Default

```
export-defaults level 2; (unless gated is configured as level 1 only)
export-defaults metric inherit;
export-defaults metric-type internal;
```

Context

isis statement

Examples

Example 1

```
isis on {
    export-defaults level 2;
    export-defaults metric 1;
    export-defaults metric-type external;
    interface ex0 cost 1;
};
```

Example 2

The following example will cause all static routes to be exported into IS-IS at level 1.

```
isis on {
    export-defaults level 1;
};
```

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

Example 3

The following example sets the default metric for routes exported into IS-IS to 10.

```
isis on {
    export-defaults metric 10;
};
```

Example 4

The following example sets the default metric type for routes exported into IS-IS to **external**.

```
isis on {
```

```
    export-defaults metric-type external;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

export on page 623

extended-metrics on page 174

isis statement on page 185

“Chapter 32 Route Exportation” on page 145 of *Configuring GateD*

rfc1195-metrics on page 205

extended-metrics

Name

extended-metrics - configures whether the router should originate extended (wide) metrics in the LSPs

Syntax

```
extended-metrics ( on | off ) ;
```

Parameters

on - enables the transmission and processing of extended metrics

off - disables the transmission and processing of extended metrics

Description

The **extended-metrics** statement configures whether or not the router shall originate and process (used in the SPF) the “Extended IP Reachability TLV” and the “Extended IS Reachability TLV.” These new TLVs (Type Length Value) allow for metrics larger than 63. Enabling extended-metrics causes GateD to originate these options in our Link State Packets (LSP) and process them in the Shortest Path First (SPF) calculation.

The setting of this option is independent of the setting of the **rfc1195-metrics** option. If both types of metrics are enabled, the lowest cost path is used in the SPF calculation.

Default

```
extended-metrics off ;
```

Context

isis statement

Examples

The following example sets GateD to originate and process extended metrics.

```
isis on {  
    extended-metrics on ;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

isis on page 185

rfc1195-metrics on page 205

external preference

Name

external preference - configures the router preference for IS-IS external routes

Syntax

```
external preference preferencenum ;
```

Parameters

preferencenum - an integer between 0 and 255

Description

The **external preference** statement sets the preference to assign to any route received in IS-IS that is marked as **external**. A route is marked as **external** when it is exported into IS-IS from another protocol via **rfc1195-metrics** with external reachability.

Each protocol on the router assigns a preference to the routes it makes available to the router. When more than one route to a specific destination exists on the router (for example, from **isis** and **bgp**) the route with the lower preference value is used by the router.

For more information on preference, see **preference** on page 199.

Default

```
external preference 151 ;
```

Context

isis statement

Examples

The following example sets the **external preference** to 140.

```
isis on {  
    external preference 140;  
};
```

See Also

"Chapter 3 Preferences and Route Selection" on page 11 of *Configuring GateD*

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

"Chapter 32 Route Exportation" on page 145 of *Configuring GateD*

preference on page 199

rfc1195-metrics on page 205

hello-interval

Name

hello-interval - configures the hello interval for a non-DIS interface

Syntax

```
hello-interval intervalnum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

intervalnum - a number of seconds between 1 and 100

Description

The **hello-interval** statement configures the rate at which hello packets are sent out an interface. IS-IS uses hello packets to learn about other routers on the link. Every *intervalnum* seconds, the router will multicast a hello to the link, announcing both itself and all the other routers it knows on the link. When the router receives a hello, it adds the sending router to its known list of routers on the link. Through this exchange, both routers learn of each other and acknowledge this fact. Until this exchange and acknowledgement happens, packets will not be forwarded between the routers.

Each hello packet also contains a hold time. The hold time informs the receiving router how long to wait without seeing another hello from the sending router before considering the sending router down. The announced hold time is derived from the router's **hello-interval** and the **hello-multiplier** as follows:

hold time = **hello-interval** * **hello-multiplier**

Setting the **hello-interval** smaller will decrease the amount of time before a router's presence (or lack thereof) is noticed, but will increase the link usage. Setting the **hello-interval** larger will decrease the link usage, but will increase the amount of time required to notice the router is up or down.

Through the use of the **level** keyword, you can configure separate intervals for each level.

Default

```
hello-interval 10 level 1 and 2 ;
```

Context

isis interface statement

Examples

Example 1

The following example sets the hello interval on interface eth0 to 5 seconds.

```
isis on {  
    interface eth0 {  
        hello-interval 5;  
    }  
}
```

```
    }  
};
```

Example 2

This example sets per-level hello intervals for interface eth0. For level 1, the interval is set to 5; for level 2, the interval is set to 8.

```
isis on {  
    interface eth0 {  
        hello-interval 5 level 1;  
        hello-interval 8 level 2;  
    }  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`dis-hello-interval` on page 164

`hello-multiplier` on page 178

`interface` on page 184

`isis` on page 185

hello-multiplier

Name

hello-multiplier - configures the hello multiplier for an interface

Syntax

```
hello-multiplier multipliernum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

multipliernum - a number between 1 and 100

Description

hello-multiplier sets the hello multiplier for an interface. The hello multiplier is used to calculate the hold time announced in hello packets as follows:

hold time = **hello-interval** * **hello-multiplier**

Each hello packet contains a hold time. The hold time informs the receiving routers how long to wait without seeing another hello from the sending router before considering the sending router down.

The hello multiplier can also be considered the number of hello packets allowed to be missing before a router will be considered down. For example, if the **hello-multiplier** is 3, then other routers must miss three consecutive hello packets before they consider this router down. Setting the **hello-multiplier** to a smaller number will decrease the amount of time before other routers notice that this router is down, but will increase the chances that a router is falsely considered down (e.g., if packets are being dropped).

Setting the hello interval to a larger number will increase the amount of time before other routers notice that this router is down, but will decrease chances that the router is falsely considered down (for example, if packets are being dropped).

Through the use of the **level** keyword, you can configure separate multipliers for each level.

Default

```
hello-multiplier 3 level 1 and 2 ;
```

Context

isis interface statement

Examples

Example 1

The following example sets the hello multiplier on interface eth0 to 5.

```
isis on {  
    interface eth0 {
```

```
        hello-multiplier 5;
    }
};
```

Example 2

This example sets per-level hello multipliers for interface eth0. For level 1, the interval is set to 4; and for level 2 the interval is set to 5.

```
isis on {
    interface eth0 {
        hello-multiplier 4 level 1;
        hello-multiplier 5 level 2;
    }
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`dis-hello-interval` on page 164

`hello-interval` on page 176

`interface` on page 184

`isis` on page 185

hold-time

Name

`hold-time` - used to set the ISO ES hold time in the operating system kernel

Syntax

```
hold-time seconds;
```

Parameters

seconds - the value of the ISO ES hold time

Description

This value is used to set the ISO ES hold time in the operating system kernel. This requires ISO socket support in the operating system.

Default

```
hold-time 120;
```

Context

`isis` statement

Examples

```
isis on {  
    hold-time 120;  
    interface ex0 cost 1;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` on page 185

hostname

Name

hostname - configures the RFC 2763 hostname

Syntax

```
hostname "name" ;
```

Parameters

name - the hostname for this IS

Description

hostname sets the value of the "Dynamic Hostname" option described in RFC 2763. This name will be used in both level 1 and level 2 Link State Packets if configured. The maximum length is 255 bytes.

Default

none

Context

isis statement

Examples

The following example configures the host name to be "foorouter".

```
isis on {  
    hostname "foorouter";  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

inet

Name

`inet` - configures IS-IS to route IPv4

Syntax

```
inet (on | off) ;
```

Parameters

none

Description

`inet` is used to specify whether or not IS-IS can exchange IPv4 reachability. `on` configures it to use IPv4; `off` configures it to not use IPv4.

Default

```
inet on ;
```

Context

`isis` statement

Examples

The following example configures IS-IS to route IPv4:

```
isis on {  
    inet on ;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`isis` on page 185

inet6

Name

`inet6` - configures IS-IS to route IPv6

Syntax

```
inet6 (on | off) ;
```

Parameters

none

Description

`inet6` is used to specify whether or not IS-IS can exchange IPv6 reachability. `on` configures it to use IPv6; `off` configures it to not use IPv6.

Note: IPv6 support must be present in GateD and in the underlying OS for this option to parse.

Default

```
inet6 off ;
```

Context

`isis` statement

Examples

The following example configures IS-IS to route IPv6:

```
isis on {  
    inet6 on ;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` on page 185

interface

Name

interface - configures IS-IS to run on the given interface

Syntax

```
interface interface_name [ { isis-interface-statements } ] ;
```

Parameters

interface_name - an interface name

isis-interface-statements - IS-IS interface-specific configuration statements

Description

interface is used to configure **isis** on an interface. Interface-specific configuration is accomplished through the use of the substatements: **auth**, **csn-interval**, **disable**, **dis-hello-interval**, **enable**, **encap**, **hello-interval**, **hello-multiplier**, **lsp-interval**, **level** (interface), **max-burst**, **mesh-group**, **metric**, **passive**, **periodic-csn**, **priority**, **retransmit-interval**.

If the **interface** *ifname* is enabled, it will be announced as reachable in the IS-IS routing domain.

Default

```
interface all;
```

Context

isis statement

Examples

The following example configures IS-IS to run on eth0 and ser0.

```
isis on {  
    interface eth0;  
    interface ser0;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

disable on page 163

enable on page 169

isis statement on page 185

isis

Name

isis - configures IS-IS

Syntax

```
isis on [ { [ isis-statements ] } ] ;  
isis off [ { [ isis-statements ] } ] ;
```

Parameters

isis-statements - **isis**-specific configuration statements, which are described throughout this chapter

Description

The **isis** statement is used to enable (**on**) or disable (**off**) IS-IS on the router. IS-IS-specific options can be configured through specific **isis** statements, which are described throughout this chapter.

Default

```
isis off;
```

Context

global

Examples

Example 1

The following configures IS-IS to run on all broadcast and IS-IS capable p2p interfaces using the default configuration.

```
isis on {  
    interface all;  
};
```

Example 2

In this example, IS-IS is enabled and configured to run on eth0 and fxp1.

```
isis on {  
    interface eth0;  
    interface fxp1;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

level

Name

level - configures the level(s) of the router or interface

Syntax

```
level ( 1 | 2 | 1 and 2 ) ;
```

Parameters

none

Description

The global level statement sets the levels at which this router is willing to operate. A level 1 router will communicate only with other level 1 routers in its area. A level 2 router will communicate with any other level 2 router, regardless of which area the other router is in. A level 1 and 2 router will communicate with other level 1 routers in its own area, and with any level 2 router, regardless of which area the other router is in. The **interface level** statement sets the levels at which this interface is willing to operate. The **interface level** must be less than or equal to the **router level**.

Level 2 routers form the layer above the level 1 areas that interconnect those areas. Level 2 must be fully connected. That is, you must not have two or more disconnected sets of level 2 routers.

When traffic in an area must reach another area, it is forwarded to a level 2 router. (How this occurs depends on the **domain-wide**, **summary-originate**, and **summary-filter** statements.) By default, a level 1 router will forward all traffic not destined for its own area to the nearest level 2 router.

If the level 2 router does not know how to reach the area for which traffic is destined, the traffic will be dropped. This is why all level 2 routers must be connected.

Default

```
level 1 and 2;
```

Context

isis statement

isis interface statement

Examples

Example 1

The following example configures the router to partake only in level 1 routing.

```
isis on {  
    level 1;  
};
```

Example 2

The following example configures the eth0 interface to partake only in level 1 routing.

```
isis on {  
    interface eth0 {  
        level 1;  
    };  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`interface` statement on page 184

`isis` statement on page 185

lsp-interval

Name

lsp-interval - time between successive transmission of a Link State Packet (LSP)

Syntax

```
lsp-interval msintervalnum ;
```

Parameters

msintervalnum - a number in milliseconds from 1 to 3000

Description

The **lsp-interval** statement configures the average time in milliseconds to wait after sending an LSP or Complete Sequence Number (CSN) packet on a broadcast link before sending another LSP or CSN packet on the same link. The actual time between transmissions is determined by the combination of the **lsp-interval** statement and the **max-burst** statement. The router is allowed to burst up to **max-burst** consecutive packets back-to-back on the link, provided that over any 1 second interval, the average time between transmissions was *msintervalnum*.

Default

```
lsp-interval 33;
```

Context

isis interface statement

Examples

The following example sets the lsp interval to 300ms.

```
isis on {  
    interface ser0 {  
        lsp-interval 300;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

max-burst statement on page 190

max-burst

Name

max-burst - configures the maximum number of link state packets (LSPs) to burst

Syntax

```
max-burst burstnum ;
```

Parameters

burstnum - a number of LSPs from 1 to 10

Description

The **max-burst** count configures the number of consecutive back-to-back transmissions of LSPs or complete sequence number (CSN) packets on a given link. The router must on average not transmit more than one LSP or CSN packet per **lsp-interval**. CPU usage can be lowered by allowing the router to actually transmit the packets in bursts. Setting *burst-num* higher will allow the router more CPU time for other actions, but may cause packets to be dropped (e.g., if other routers cannot handle the back-to-back traffic).

Default

```
max-burst 5;
```

Context

isis interface statement

Examples

Example 1

The following example configures the maximum burst of 10 on interface eth0.

```
isis on {  
    interface eth0 {  
        max-burst 10;  
    };  
};
```

Example 2

This example effectively disables bursting by restricting it to 1 on interface eth0.

```
isis on {  
    interface eth0 {  
        max-burst 1;  
    };  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

lsp-interval statement on page 189

mesh-blocked

Name

mesh-blocked - prevents the flooding of Link State Packets (LSPs) on an interface

Syntax

```
mesh-blocked ;
```

Parameters

none

Description

The **mesh-blocked** statement prevents LSPs from being flooded out on an interface at certain times. It is most useful for pruning flooding topologies in a full mesh network topology.

Default

none

Context

isis interface statement

Examples

The following example prevents LSPs from being flooded out the eth0 interface.

```
isis on {  
    interface eth0 {  
        mesh-blocked;  
    };  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

mesh-group on page 193

mesh-group

Name

mesh-group - configures the mesh group number for an interface

Syntax

```
mesh-group meshnum ;
```

Parameters

meshnum - the mesh group number for this interface

Description

mesh-group implements an extension to the IS-IS protocol described in RFC 2973 which allows an IS to track a "mesh group" number that prevents redundant flooding when a full mesh of circuits are used among a group of ISs. Use of this option may be combined with "**mesh-blocked**" on other circuits to gain optimal flooding topologies. This may be any 32-bit value including 0.

Default

none

Context

isis interface statement

Examples

The following example sets the mesh group of interface eth0 to be 1.

```
isis on {  
    interface eth0 {  
        mesh-group 1;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

mesh-blocked statement on page 192

metric

Name

metric - sets the outbound cost of the IS-IS interface at the given level

Syntax

```
metric metricnum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

metricnum - a number from 1 to *max_metric*. *max_metric* is 63 if **rfc1195-metrics** is on; otherwise it is $2^{24}-1$ (or 4,261,412,864).

Description

The **metric** statement sets the interface outbound metric to *metricnum*. This value is advertised throughout the IS-IS routing domain as the cost to send a packet out the interface, and is used to calculate the shortest path between routers.

Each interface is capable of having up to two unique metric values, one for **level 1** and one for **level 2**. If a level is not specified, it defaults to setting both the **level 1 and level 2** metric to the indicated value *metricnum*.

Default

```
metric 10 level 1 and 2;
```

Context

isis interface statement

Examples

Example 1

The following will set interface eth0's metric to 4.

```
isis on {  
    interface eth0 {  
        metric 4;  
    };  
};
```

Example 2

This example sets the **level 1** metric of eth0 to 1 and the **level 2** metric to 5.

```
isis on {  
    interface eth0 {  
        metric 1 level 1;  
    };  
};
```

```
metric 5 level 2;  
};  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

extended-metrics on page 174

interface statement on page 184

isis statement on page 185

rfc1195-metrics on page 205

overload-bit

Name

`overload-bit` - administratively set the overload bit

Syntax

```
overload-bit ( on | off ) ;
```

Parameters

none

Description

The `overload-bit` statement configures whether the router announces itself as overloaded to the IS-IS routing domain. If the `overload-bit` is `on`, then no traffic will be directed at the router for forwarding; however, traffic originated by (or received at) the router will still be forwarded properly. This can be useful in testing configurations or monitoring IS-IS operation without actually affecting it. It can also be used to temporarily disable the router for diagnostics.

Default

```
overload-bit off;
```

Context

`isis` statement

Examples

The following example configures the router to announce itself as being overloaded.

```
isis on {  
    overload-bit on;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`isis` statement on page 185

passive

Name

passive - configures an interface to be passive

Syntax

```
passive ( on | off ) ;
```

Parameters

none

Description

The **passive** statement configures whether the interface actively participates in IS-IS on the given link. If **passive** is set to **on**, the link will be advertised in the router's link state packet (LSP) and thus enable routing to that link through the router; however, the router will not actually run IS-IS on the link. This has the effect of allowing traffic destined to the link's networks to use this router but not allowing traffic destined for other networks to use this router and link to reach those networks.

Default

```
passive off;
```

Context

isis interface statement

Examples

The following example configures interface eth0 to be **passive**.

```
isis on {  
    interface eth0 {  
        passive on;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

periodic-csn

Name

periodic-csn - configures periodic Complete Sequence Number (CSN) flooding on point-to-point interfaces

Syntax

```
periodic-csn ( on | off ) ;
```

Parameters

none

Description

In typical configurations, it is not necessary or desirable to send CSNs at regular intervals on point-to-point links. These packets are normally transmitted only upon initial establishment of the adjacency. This option forces CSN packets to be sent on an interface, subject to the normal CSN control parameters such as **csn-interval**, **max-burst**, and **lsp-interval**. This option makes sense only on point-to-point links. To block CSN packet flooding on LAN interfaces, see the **passive** option.

Default

```
periodic-csn off;
```

Context

isis interface statement

Examples

The following example forces periodic CSN flooding on interface gif0.

```
isis on {  
    interface gif0 {  
        periodic-csn on;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

mesh-blocked statement on page 192

mesh-group statement on page 193

passive statement on page 197

preference

Name

preference - configures the router preference for IS-IS internal routes

Syntax

```
preference preferencenum ;
```

Parameters

preferencenum - a number from 0 to 255

Description

The **preference** statement sets the **preference** to assign to any route received in IS-IS that is marked as internal. All networks in the IS-IS routing domain are marked as internal. If **rfc1195-metrics** is on, then routes exported into IS-IS from other protocols are marked as external and will not be affected by this statement, even those marked by an internal metric type.

Each protocol on the router assigns a **preference** to the routes it makes available to the router. When more than one route to a specific destination exists on the router (for example, from **isis** and **rip**) the route with the lower **preference** value is used by the router.

Default

```
preference 11;
```

Context

isis statement

Examples

The following example sets the **preference** assigned to internal IS-IS routes to 5.

```
isis on {  
    preference 5;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

export on page 623

external preference on page 175

priority

Name

priority - configures the priority of an interface for Designated Intermediate System (DIS) election

Syntax

```
priority prioritynum [ level ( 1 | 2 | 1 and 2 ) ] ;
```

Parameters

prioritynum - a number from 0 to 127

Description

The **priority** statement configures the priority of this router to become the DIS for the given interface's link. The router with the highest **priority** on a link will be elected DIS. If more than two routers have the same highest priority, the interface's MAC address breaks the tie (highest winning).

The DIS for a link is ultimately responsible for making sure all other routers on the same link have the same link state database. It is therefore important that the router elected DIS be stable. The DIS will be required to process more IS-IS routing packets, and should thus be capable of doing so.

Through the use of the **level** keyword, one can configure separate priorities for each level.

Default

```
priority 64 level 1 and 2;
```

Context

isis interface statement

Examples

The following example configures interface eth0's priority to be 0, thus making it as unlikely as possible to be elected DIS.

```
isis on {  
    interface eth0 {  
        priority 0;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`csn-interval` statement on page 161

`dis-hello-interval` statement on page 164

`isis` statement on page 185

psn-interval

Name

psn-interval - configures the rate of Partial Sequence Number (PSN) transmission

Syntax

```
psn-interval intervalnum ;
```

Parameters

intervalnum - a number of seconds from 1 to 20

Description

The **psn-interval** statement sets the rate at which PSN packets are multicast from the IS on broadcast interfaces.

On point-to-point links, PSN packets are used to acknowledge receipt of Link State Packets (LSPs.) If an LSP is received, but a PSN acknowledgement is not sent, the LSP will be retransmitted by the sending router. We recommend that the **psn-interval** be set smaller than **retransmit-interval**.

On broadcast links, non-DIS circuits use PSN packets to request newer LSPs that have been advertised by the DIS in CSN packets. See **csn-interval** for more description of CSN packets.

Default

```
psn-interval 2;
```

Context

isis statement

Examples

The following example configures the **psn-interval** to be 1 second.

```
isis on {  
    psn-interval 1;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

csn-interval on page 161

isis statement on page 185

retransmit-interval on page 204

require-snp-auth

Name

`require-snp-auth` - determines whether SNP authentication is required

Syntax

```
require-snp-auth ( on | off ) ;
```

Parameters

none

Description

`require-snp-auth` sets whether Complete Sequence Number (CSN) and Partial Sequence Number (PSN) packets should be required to contain authentication.

The IS-IS standard requires that all IS-IS packets contain authentication (if the router is configured with authentication); however, some routers do not conform to this standard regarding CSN and PSN packets.

The default is to not require authentication of CSN and PSN packets, so that the router will interoperate with these other vendors.

Default

```
require-snp-auth off;
```

Context

`isis` statement

Examples

The following example enables requiring authentication of CSN and PSN packets.

```
isis on {  
    require-snp-auth on;  
};
```

See Also

`area-auth` on page 155

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

`domain-auth` on page 166

`isis` statement on page 185

retransmit-interval

Name

retransmit-interval - time before retransmitting an LSP on a point-to-point interface

Syntax

```
retransmit-interval intervalnum ;
```

Parameters

intervalnum - a number of seconds from 1 to 100

Description

The **retransmit-interval** sets the amount of time to wait between successive transmissions of the same LSP on a point-to-point link. Setting this value higher will avoid unnecessary retransmission, but will slow convergence (synchronization) time between routers. Setting this value lower will cause convergence time to decrease, but may cause unnecessary retransmission (e.g., the other router has received the LSP initially but not had a chance to acknowledge the fact). This number should be longer than the **psn-interval**.

Default

```
retransmit-interval 5;
```

Context

isis interface statement

Examples

The following example configures the retransmit interval to be 10 seconds.

```
isis on {  
    interface ser0 {  
        retransmit-interval 10;  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

interface statement on page 184

isis statement on page 185

psn-interval on page 202

rfc1195-metrics

Name

rfc1195-metrics - configure whether the router should originate metrics in the normal IS Neighbors Type Length Value (TLV)

Syntax

```
rfc1195-metrics ( on | off ) ;
```

Parameters

none

Description

There are two IS-IS options that may be used to originate reachability to neighbors. The option specified in RFC 1195 allows a metric no larger than 63. This parameter indicates whether the router originates metrics using this type of option.

Default

```
rfc1195-metrics on ;
```

Context

isis statement

Examples

```
isis on {  
    rfc1195-metrics on;  
    interface all {  
        enable;  
    };  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

extended-metrics statement on page 174

isis statement on page 185

ribs

Name

ribs - configures the Routing Information Bases (RIBs) into which IS-IS installs routes

Syntax

```
ribs ( unicast | unicast multicast ) ;
```

Parameters

none

Description

ribs configures the RIBs into which IS-IS should install its. If the router has multicast capability, IS-IS can be configured to install routes in the multicast RIB as well as the unicast RIB. IS-IS cannot be configured to install routes into only the multicast RIB. The multicast RIB is used by protocols such as **pim**.

Default

```
ribs unicast;
```

Context

isis statement

Examples

The following example configures the router to install IS-IS routes in both the unicast and multicast RIBs.

```
isis on {  
    ribs unicast multicast;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

"Chapter 25 Multicast Statement" on page 123 of the *Configuring GateD Guide*

isis statement on page 185

multicast statement on page 545

spf-interval

Name

spf-interval - configures the wait time before calculating the routing table

Syntax

```
spf-interval intervalnum ;
```

Parameters

intervalnum - a number of seconds from 1 to 60

Description

spf-interval sets the amount of time to wait in seconds before running the routing table calculation after it first determines that it needs to do so.

When new Link State Packets (LSPs) are received or originated by the router, the routing table needs to be recalculated. It would be inefficient to run the routing table calculation every time a new LSP was received or originated, because many LSPs may be being received or originated quickly. The **spf-interval** guarantees that the calculation will not occur more often than *intervalnum* seconds.

When a new LSP is received or originated, a timer is set to expire in *intervalnum* seconds. When it expires, the routing table is recalculated. During this wait time, any new LSPs received or originated will also be processed in the forthcoming calculation.

Setting this value higher will decrease the amount of CPU time the router spends calculating the routing table but will slow down the convergence time. Setting this value lower will increase convergence time at the cost of CPU usage.

Default

```
spf-interval 2;
```

Context

isis statement

Examples

The following example configures the spf interval to be 5 seconds.

```
isis on {  
    spf-interval 5;  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

start-accept

Name

start-accept - configures the time at which an MD5 key shall start to be accepted by this router

Syntax

```
start-accept YYYY/MM/DD HH:MM [.ss] ;
```

Parameters

YYYY/MM/DD HH:MM [.ss] ; - a date/time specification

Description

This command configures the time at which an MD5 key shall start to be accepted by this router.

Default

The default is to always accept the key.

Context

isis domain auth md5 statement

isis area auth md5 statement

isis interface auth md5 statement

Examples

```
isis on {  
    area auth {  
        md5 key "foo" {  
            start-accept 2001/12/1 10:15.22;  
        };  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

auth on page 157

area auth on page 160

domain auth on page 166

start-generate on page 210

stop-accept on page 212

stop-generate on page 214

start-generate

Name

start-generate - configures the time at which an MD5 key shall start to be generated by this router

Syntax

```
start-generate YYYY/MM/DD HH:MM [.ss] ;
```

Parameters

YYYY/MM/DD HH:MM [.ss] ; - a date/time specification

Description

This command configures the time at which an MD5 key shall start to be generated by this router. Note that the expiration of a start-generate time does not cause automatic regeneration of the Link State Packets (LSPs); the new authentication will be originated when the LSPs are originated for some other reason. At longest, this is one refresh interval.

Default

The default is to always generate the key. When more than one key is configured for a time range, configuration file order breaks ties.

Context

```
isis domain auth md5 statement  
isis area auth md5 statement  
isis interface auth md5 statement
```

Examples

```
isis on {  
    area auth {  
        md5 key "foo" {  
            start-generate 2001/12/1 10:15.22;  
        };  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

auth on page 157

`area auth` on page 160
`domain auth` on page 166
`start-accept` on page 208
`stop-accept` on page 212
`stop-generate` on page 214

stop-accept

Name

stop-accept - configures the time at which an MD5 key shall stop being accepted by this router

Syntax

```
stop-accept YYYY/MM/DD HH:MM [.ss] ;
```

Parameters

YYYY/MM/DD HH:MM [.ss] ; - a date/time specification

Description

This command configures the time at which an MD5 key shall stop being accepted by this router.

Default

The default is to always accept the key.

Context

```
isis domain auth md5 statement  
isis area auth md5 statement  
isis interface auth md5 statement
```

Examples

```
isis on {  
    area auth {  
        md5 key "foo" {  
            stop-accept 2001/12/1 10:15.22;  
        }  
    }  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

auth on page 157

area auth on page 160

domain auth on page 166

start-accept on page 208

start-generate on page 210

stop-generate on page 214

stop-generate

Name

stop-generate - configures the time at which an MD5 key shall cease to be generated by this router

Syntax

```
stop-generate YYYY/MM/DD HH:MM [.ss] ;
```

Parameters

YYYY/MM/DD HH:MM [.ss] ; - a date/time specification

Description

This command configures the time at which an MD5 key shall cease to be generated by this router.

Default

The default is to always generate the key.

Context

```
isis domain auth md5 statement  
isis area auth md5 statement  
isis interface auth md5 statement
```

Examples

```
isis on {  
    area auth {  
        md5 key "foo" {  
            stop-generate 2001/12/1 10:15.22;  
        };  
    };  
};
```

See Also

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

auth on page 157

area auth on page 160

domain auth on page 166

start-accept on page 208

start-generate on page 210

stop-accept on page 212

summary-filter

Name

summary-filter - configures filtering of downward summarization

Syntax

```
[ summary-filter [ inet ] {  
    [ ipv4-network mask ipv4-netmask ; ]  
    [ ipv4-network masklen ipv4-masklen ; ]  
} ; ]  
[ summary-filter inet6 {  
    [ ipv6-network mask ipv6-netmask ; ]  
    [ ipv6-network masklen ipv6-masklen ; ]  
} ; ]
```

Parameters

inet - specifies that the **summary-filter** statement applies to IPv4

inet6 - specifies that the **summary-filter** statement applies to IPv6

ipv4-network - an IPv4 address in dotted-quad format

ipv4-netmask - an IPv4 **mask** in dotted-quad format

ipv4-masklen - a number from 0 to 32

ipv6-network - an IPv6 address in colon-separated format

ipv6-netmask - an IPv6 **mask** in colon-separated format

ipv6-masklen - a number from 0 to 128

Description

summary-filter configures filters to block advertisement of networks from level 2 into level 1. It is used only when **domain-wide** is configured to be **on**.

If a level 1 and 2 router is configured with **domain-wide on**, each level 2 route is by default advertised down into level 1. This advertisement may be restricted by using **summary-filter**. If a level 2 route that would normally be announced down into level 1 matches an address and mask pair in the **summary-filter** statement, that announcement is blocked.

Default

no filtering

Context

isis statement

Examples

Example 1

The following example blocks announcement of any level 2 route to 32.0.0.0/8 or more specific (e.g., 32.1.0.0/16) into level 1.

```
isis on {
    domain-wide on;
    summary-filter {
        32.0.0.0 masklen 8;
    };
};

isis on {
    inet6 on;
    summary-filter inet6 {
        fec0:0:0:031f::e masklen 16;
    };
};
```

Example 2

This example blocks all announcements of level 2 routes into level 1.

```
isis on {
    domain-wide on;
    summary-filter {
        0.0.0.0 masklen 0;
    };
};

isis on {
    inet6 on;
    summary-filter inet6 {
        :: masklen 0;
    };
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`domain-wide` statement on page 168

isis statement on page 185

summary-originate on page 219

summary-originate

Name

summary-originate - configures summarization for level 1 and 2 routers

Syntax

```
[ summary-originate [ inet ] {
    [ ipv4-network mask ipv4-netmask metric cost-value ; ]
    [ ipv4-network masklen ipv4-masklen metric cost-value ; ]
} ; ]
[ summary-originate inet6 {
    [ ipv6-network mask ipv6-netmask metric cost-value ; ]
    [ ipv6-network masklen ipv6-masklen metric cost-value ; ]
} ; ]
```

Parameters

inet - specifies that the **summary-originate** statement applies to IPv4

inet6 - specifies that the **summary-originate** statement applies to IPv6

ipv4-network - an IPv4 address in dotted-quad format

ipv4-netmask - an IPv4 mask in dotted-quad format

ipv4-masklen - a number from 0 to 32

cost-value - a number from 1 to *max_metric*

ipv6-network - an IPv6 address in colon-separated format

ipv6-netmask - an IPv6 mask in colon-separated format

ipv6-masklen - a number from 0 to 128

Description

summary-originate controls which reachable IP networks are advertised from level 1 to level 2 by a level 1 and 2 router.

When a level 1 and 2 router has a route in its level 1 area, it by default announces that route into level 2. To reduce the size of the level 2 database (i.e., the number of routes advertised into level 2), a router can configure aggregating **summary-originate** statements.

If a level 1 route exists that matches an address and mask pair in the **summary-originate** statement, the following actions occur:

The level 1 route is not advertised in level 2. If the matching address and mask pair has a configured metric, the address and mask pair is announced into level 2 with that metric; otherwise, the matching address and mask pair are marked **restrict** and no advertisement into level 2 will occur.

Use of the **restrict** keyword can, for example, allow one to re-use private addresses within level 1 areas.

max_metric is 63 if **rfc1195-metrics** is on; otherwise it is $2^{24}-1$ (or 4,261,412,864).

Default

no summarization

Context

isis statement

Examples

Example 1

```
isis on {  
    summary-originate {  
        32.0.0.0 masklen 8 metric 1;  
    };  
};
```

Example 2

```
isis on {  
    summary-originate {  
        32.0.0.0 masklen 8 metric 1;  
        10.0.0.0 masklen 0 restrict;  
    };  
};
```

Example 3

```
isis on {  
    inet6 on;  
    summary-originate inet6 {  
        fec0:0:0:031f::e masklen 64 metric 1;  
    };  
};
```

Example 4

```
isis on {  
    inet6 on;  
    summary-originate inet6 {  
        3ffd:0:0:abcd:F800:: masklen 69 metric 1;  
        fec0:0:0:031f::e masklen 16 restrict;  
    };  
};
```

```
};  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

isis statement on page 185

summary-filter on page 216

systemid

Name

systemid - configures the router's system ID

Syntax

```
systemid D.D.D.D ;  
systemid HHHH.HHHH.HHHH ;
```

Parameters

D - a decimal number from 0 to 255

H - a hexadecimal digit from 0 to F

Description

systemid sets the system ID of the router. In OSI, each router (IS) has a network entity title (NET) assigned to it. The NET uniquely identifies the IS throughout the routing domain and determines which area the router is in. A NET is divided into the area portion, then the system ID, and is completed by a single 0 octet (byte). The system ID is always (by convention) 6 octets and the area portion is variable and can be from 1 to 13 octets. Some routers require the user to specify the area and system ID by setting the NET directly.

This mechanism is largely outdated because most users do not actually run the OSI routing domain, and it has thus been simplified to be more like other current routing protocols such as **ospf**. The **systemid** *D.D.D.D* form of the command is supplied to be interchangeable with **router-id**. It takes as its argument a dotted-quad (IPv4-like) address.

This form will set the **systemid** to be two octets of 0 followed by four octets as given by the four decimal numbers in the dotted-quad argument. The more general form of the command utilizes hexadecimal digits and is 6 octets in length. Each pair of hexadecimal digits is a single octet, and thus 12 hex-digits must be given. If a **router-id** is being specified for the router, and this is acceptable as the system ID also, you do not need to specify the **systemid**.

Default

defaults to the configured **router-id** with 2 octets of 0 prepended

Context

isis statement

Examples

Example 1

The following example sets the **systemid** to (in hex format) "0000.0000.0001", or 5 octets of zero followed by 1 octet of 1.

```
isis on {
```

```
    systemid 0.0.0.1;  
};
```

Example 2

This example sets the **systemid** to (in hex format) "0000.0000.001F" or 5 octets of zero followed by 1 octet of decimal 31.

```
isis on {  
    systemid 0000.0000.001F;  
};
```

See Also

area statement on page 153

"Chapter 13 Intermediate System to Intermediate System (IS-IS)" on page 55 of *Configuring GateD*

isis statement on page 185

traceoptions

Name

traceoptions - configures traceoptions specifically for IS-IS

Syntax

```
traceoptions std-traceoptions [ isis-traceoptions ] ;  
traceoptions [ std-traceoptions ] isis-traceoptions ;
```

Parameters

std-traceoptions - standard trace options

isis-traceoptions - The following additional options are available: **adjacency**, **dis-election**, **db**, **flood**, **spf**, **debug**. The following packet types can be traced: **csn**, **hello**, **lsp**, **packets**, **psn**.

Description

The **traceoptions** command configures what type of IS-IS specific tracing should occur. For a general discussion of the traceoptions format, see "Chapter 2 Trace Statements and Global Options" on page 3. The specific options available are as follows:

- adjacency** - Trace changes to the adjacency database.
- dis-election** - Trace changes in which router is elected DIS.
- db** - Trace changes to the LSP database.
- flood** - Trace flooding operations.
- spf** - Trace the routing table operations.
- debug** - Trace in much detail all aspects of the protocol's operation.

The packet types that can be traced are as follows:

- csn** - Complete Sequence Numbers (CSN) packets
- hello** - IS-IS Hello (IIH) packets
- lsp** - Link State Packets (LSPs)
- packets** - all packets
- psn** - Partial Sequence Numbers (PSNs) packets

If the detail flag is specified, the entire packet will be traced; otherwise, just the fixed header will be traced.

Note: Full tracing (including debugging) can be quite expensive and may affect the overall performance of the router.

Default

```
traceoptions none;
```

Context

isis statement

Examples

Example 1

The following example configures the router to trace all adjacency and SPF-related changes.

```
isis on {  
    traceoptions adjacency spf;  
};
```

Example 2

This example configures the router to trace all received and sent LSPs in full detail (i.e., including variable length fields, or TLVs).

```
isis on {  
    traceoptions detail lsp packets;  
};
```

See Also

“Chapter 13 Intermediate System to Intermediate System (IS-IS)” on page 55 of *Configuring GateD*

`isis` statement on page 185

`traceoptions` on page 3

Chapter 10

Border Gateway Protocol (BGP)

allow

Name

allow - permits peer connections from any addresses in the specified range of network and mask pairs

Syntax

```
[inet6] allow {  
    all ;  
    | host ipaddress ;  
    | classful_network ;  
    | network mask mask ;  
    | network masklen masklennumber ;  
    | network / masklennumber ;  
}
```

Parameters

classful_network - the address of the network to be peered, in dotted-quad format (xxx.xxx.xxx.xxx)

network - the address of the network to be peered, in dotted-quad format (xxx.xxx.xxx.xxx)

mask - the address mask (modification) in dotted-quad format (xxx.xxx.xxx.xxx)

masklennumber - the number of contiguous one bits at the beginning of the mask

ip_address - the address of the host network, in dotted-quad format (xxx.xxx.xxx.xxx)

Description

allow permits peer connections from addresses in the specified range of networks. Multiple networks may be specified in the **allow** clause. All parameters for these peers must be configured in the **group** clause. The internal peer structures are created when an incoming open request is received, and destroyed when the connection is broken. For more details on specifying the network/mask pairs, see "Route Filtering" on page 129 of *Configuring GateD*.

Default

none

Context

```
bgp group type external statement
bgp group type internal statement
bgp group type routing statement
```

Examples

Example 1

This allows anyone to peer with us and is not recommended.

```
allow {
    all;
} ;
```

Example 2

This allows a specific host to peer with us. Note that this is equivalent to specifying a **peer** statement without any options.

```
allow {
    host 192.0.2.1 ;
} ;
```

Example 3

This allows all hosts within a classful subnet.

```
allow {
    10 ; # Allow all 10/8
} ;
```

Example 4

This allows all hosts within a network.

```
allow {
    192.0.2.0 mask 255.255.255.0 ;
};
```

which is the same as:

```
allow {
    192.0.2.0 masklen 24 ;
} ;
```

which is the same as:

```
allow {
```

```
    192.0.2.0/24 ;  
} ;
```

Example 5

You can specify more than one network in an **allow** clause.

```
allow {  
    192.0.2.0/24 ;  
    10.0.0.0/8 ;  
    host 172.16.0.1 ;  
} ;
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*

group type on page 247

ascount

Name

ascount - configures the number of times that this router will prepend its AS number to a route's autonomous system (AS) Path when it sends the route to an external peer

Syntax

ascount *count*

Parameters

count - an integer between 1 and 25, inclusive

Description

ascount configures the number of times that this router will prepend its AS number to a route's AS Path when it sends the route to an external peer. Larger values are typically used to bias upstream peers' route selection. All things being equal, most routers will prefer routes with shorter AS Paths. Using **ascount**, the AS Path this router sends can be artificially lengthened.

The AS number that is prepended to the AS Path is the **localas** value, if specified; otherwise, it is the global **autonomoussystem** value.

ascount supersedes the **nov4asloop** option. Regardless of whether **nov4asloop** is set, this router will still send multiple copies of its own AS if the **ascount** option is set to something greater than 1.

Default

ascount 1 ;

Context

bgp group type external statement

bgp peer statement

Examples

Example 1

The **ascount** is specified for a group.

[**autonomoussystem** and **routerid** omitted]

```
bgp on {  
    group type external peeras 64512 ascount 5 {  
        peer 192.0.2.2 ; # peer inherits ascount of 5 from group  
        peer 192.0.2.3 ; # peer inherits ascount of 5 from group  
    } ;  
} ;
```

[import and export statements omitted]

Example 2

The **ascount** is specified for a given peer.

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.2 ascount 5 ; # This peer has an ascount of 5  
        peer 192.0.2.3 ;           # This peer defaults to ascount 1  
    } ;  
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*

localas on page 260

bgp

Name

bgp - enables or disables BGP

Syntax

```
bgp ( on | off ) [ { bgp parameters } ] ;
```

Parameters

bgp parameters - includes all the parameters in the BGP section of this manual

Description

bgp enables or disables BGP. *bgp parameters* includes all the parameters in the BGP section of this manual.

Default

```
bgp off ;
```

Context

global

Examples

Example 1

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1;  
    };  
};
```

Example 2

BGP can be disabled and still have everything configured.

[autonomoussystem and routerid omitted]

```
bgp off {  
    group type external peeras 64512 {  
        peer 192.0.2.1;  
    };  
};
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

clusterid

Name

clusterid - specifies the route reflection cluster ID for BGP

Syntax

```
clusterid host-id ;
```

Parameters

host-id - a router ID, in dotted-quad format (xxx.xxx.xxx.xxx), used by route reflectors to prevent route propagation loops within the cluster

Description

clusterid specifies the route reflection cluster ID for BGP. (See "Route Reflection" on page 71 of *Configuring GateD* for more information about route reflection.) The cluster ID defaults to be the same as the router ID. (See "Router ID Syntax," on page 30 of *Configuring GateD* for more information about router ID.) If a router is to be a route reflector, then a single cluster ID should be selected and configured on all route reflectors in the cluster. The only constraints on the choice of cluster ID are the following:

- IDs of clusters within an Autonomous System (AS) must be unique within that AS.
- The cluster ID must not be 0.0.0.0.

Choosing the cluster ID to be the router ID of one router in the cluster will always fulfill these criteria. If there is only one route reflector in the cluster, the **clusterid** setting may be omitted, because the default will suffice.

Default

the globally configured **routerid**

Context

bgp

Router ID Syntax

Route Reflection

Examples

Example 1

The **clusterid** is specified.

[**autonomoussystem** omitted]

```
routerid 192.0.2.1 ;
```

```
bgp on {
```

```
    clusterid 192.0.2.254 ;
```

```
    group type internal peers 65534 {
```

```

        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
# Routes received from clients will have the clusterid of
# 192.0.2.254 added to their BGP attributes.
group type internal peeras 65534 reflector-client {
    peer 192.0.2.10 ;
    peer 192.0.2.11 ;
    peer 192.0.2.12 ;
    peer 192.0.2.13 ;
} ;
} ;
[import and export statements omitted]

```

Example 2

The `clusterid` is omitted.

[autonomoussystem omitted]

```

routerid 192.0.2.1 ;
bgp on {
    group type internal peeras 65534 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
# Routes received from clients will have the clusterid of
# 192.0.2.1 added to their BGP attributes.
    group type internal peeras 65534 reflector-client {
        peer 192.0.2.10 ;
        peer 192.0.2.11 ;
        peer 192.0.2.12 ;
        peer 192.0.2.13 ;
    } ;
} ;
[import and export statements omitted]

```

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

"Route Reflection" on page 71 of *Configuring GateD*

comm

Name

comm - specifies the community attributes added to routes sent to routers in a peer group

Syntax

```
comm { community_values }
```

Parameters

community_values include:

comm-hex *hex-number hex-number* ; - This parameter specifies any arbitrary community that is the concatenation of the two sixteen-bit numbers specified.

comm-split *autonomous_system community-id* ; - This parameter specifies a community that is the concatenation of the AS number *autonomous_system* and the arbitrary sixteen-bit number, *community-id*.

community no-export ; - Specifies the well-known community NO_EXPORT as defined in RFC 1997. Routes tagged with this community are not to be exported outside of a confederation boundary.

community no-advertise ; - Specifies the well-known community NO_ADVERTISE as defined in RFC 1997. Routes tagged with this community are not to be advertised to any other peers.

community no-export-subconfed ; - Specifies the well-known community NO_EXPORT_SUBCONFED as defined in RFC 1997. Routes tagged with this community are not to be advertised to external peers, even if they are within the same confederation.

Description

comm specifies the community attributes added to routes sent to routers in a peer group. Communities can also be manipulated on import and export of routes from peers. See "BGP Communities" on page 77 of *Configuring GateD* for more information.

Default

none

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

Examples

Example 1

Add the no-export community to a group.

[autonomoussystem and routerid omitted]

```
bgp on {
    group type external peeras 65534
    comm { community no-export ; } {
        peer 192.0.2.1;
    } ;
} ;
```

[import and export statements omitted]

Example 2

Add the community 64512:100 to a group.

[autonomoussystem and routerid omitted]

```
bgp on {
    group type external peeras 65534
    comm { comm-split 64512 100 ; } {
        peer 192.0.2.1;
    } ;
} ;
```

[import and export statements omitted]

Example 3

Add the community 64512:100 and community no-export-subconfed to a group.

[autonomoussystem and routerid omitted]

```
bgp on {
    group type external peeras 65534
    comm { comm-split 64512 100 ; community no-export-subconfed ; } {
        peer 192.0.2.1;
    } ;
} ;
```

[import and export statements omitted]

See Also

"BGP Communities" on page 77 of *Configuring GateD*

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

export on page 623

import on page 614

confed

Name

confed - marks a BGP group as being part of a BGP confederation

Syntax

confed

Parameters

none

Description

confed configures a BGP group to be part of a BGP confederation. The **confed-id** keyword must have been previously specified. When the **confed** keyword is present on a BGP group statement, GateD will treat all members of that group as confederation peers. Additionally, BGP will use the configured **autonomoussystem** number in its peering session for its AS number instead of **confed-id**.

When sending UPDATES to confederation peers, the AS_PATH is modified using AS_CONFED_SET and AS_CONFED_SEQUENCES instead of the normal CONFED_SET and CONFED_SEQUENCE. Additionally, the restriction against propagating MED and LOCAL_PREF values is relaxed and these values are propagated to confederation external peers.

Default

off

Context

bgp group statement

Examples

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 propagate everything from our confederation.
export bgp as 200 {
    proto bgp as 64512 {
        all ;
    } ;
    proto bgp as 65000 {
        all ;
    } ;
} ;
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

RFC 3065 - BGP Autonomous System Confederations at

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

defaultmetric

Name

defaultmetric - defines the metric Multi-Exit Discriminator (MED) used when advertising routes via BGP

Syntax

```
defaultmetric metric ;
```

Parameters

metric

Description

defaultmetric defines the metric (MED) used when advertising routes via BGP. If not specified, no metric is propagated. This metric may be overridden by a metric specified on the **peer** or **group** statements, or in export policy.

Default

none

Context

bgp

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    defaultmetric 100 ;
    group type external peeras 64512 {
# Metric of 100 is sent to this peer by default.
        peer 192.0.2.2 ;
# Metric of 150 is sent to this peer.
        peer 192.0.2.3 metricout 150 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

"MED Configuration Example" on page 68 of *Configuring GateD*

metricout on page 267

discard-nonprefixed-confederations

Name

discard-nonprefixed-confederations - discards malformed AS_PATHs containing non-prefixed confederation segments

Syntax

discard-nonprefixed-confederations

Parameters

none

Description

discard-nonprefixed-confederations causes BGP to discard AS_PATHs that are received from BGP peers where there are BGP Confederation AS_PATH segments (AS_CONFED_SEQUENCE, AS_CONFED_SET) occurring anywhere other than at the left hand side of the AS_PATH. This feature is useful due to "buggy" routers on the Internet that will illegally advertise AS_PATHs containing confederation segments outside of a confederation boundary. If these routes propagate beyond the confederation boundary edge, they will cause the peering session of any router that does not accept confederation segments from non-confederation peers to drop the peering session and thus disrupt service.

This option will cause routes containing non-prefixed confederation segments to be logged and discarded.

Default

not enabled

Context

bgp statement

Examples

```
routerid 192.0.0.1;
autonomoussystem 64512;

bgp on {
    discard-nonprefixed-confederations;

    group type external peeras 65534 {
        allow {
            all;
        };
    };
};
```

```
} ;
```

[`static`, `import`, and `export` statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

`ignore-nonprefixed-confederations` on page 254

RFC 3065 - Autonomous System Confederations for BGP

gateway

Name

gateway - instructs GateD to use a form of multihop external border gateway protocol (EBGP)

Syntax

gateway *host*

Parameters

host - A gateway is an intermediate destination by which packets are delivered to their ultimate destination. A gateway is the IP address of any host.

Description

gateway instructs GateD to use a form of multihop EBGP. If a network is not shared with this group, the **gateway** *host* specifies a router on an attached network to be used as the next hop router for routes received from this peer. The **gateway** parameter can also be used to specify a next hop for groups that are on shared networks. For example, you might use **gateway** to ensure that third-party next hops are never accepted from a given group by specifying that group's address as its own gateway. The gateway specified must have consistent routing information to prevent routing loops. **gateway** is not needed in most cases.

Default

none

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

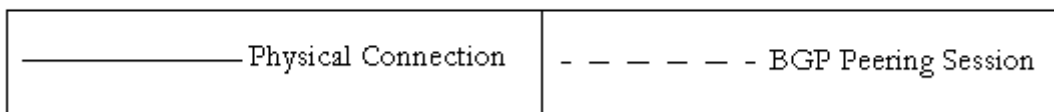
bgp peer statement

Examples

Consider the following example:

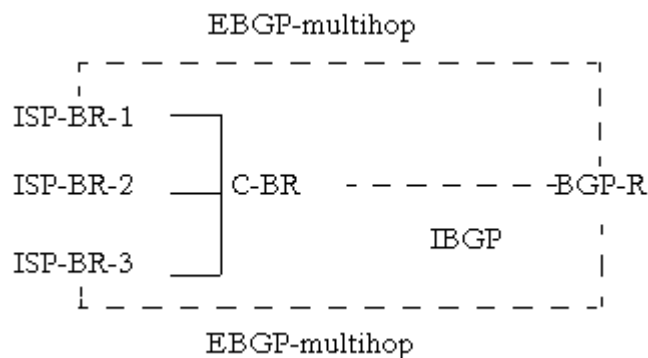
THE ROUTE SIEVE

Key:



A company has three Internet feeds. The customer border router has direct connections to the ISP border routers. However, the company's router does not have enough resources to hold three full views of the Internet. The company uses a GateD workstation running EBGP multihop

to establish peering sessions with the ISP border routers and uses IBGP to send the selected routes to the EBG-multi-hop company border router.



Router	AS	RouterID/Interface
ISP-BR-1	65501	192.168.1.1 -- DS3 to C-BR
ISP-BR-2	65502	10.0.0.1 -- DS3 to C-BR
ISP-BR-3	65503	172.16.0.1 -- DS3 to C-BR
C-BR	64512	192.0.2.1 -- Gig Ethernet to BGP-R
BGP-R	64512	192.0.2.2 -- Gig Ethernet to C-BR

Example 1

Configure BGP-R in the **group** statement.

```

autonomoussystem 64512 ;
routerid 192.0.2.1 ;
bgp on {
    group type external peeras 65501 gateway 192.168.1.1 {
        peer 192.168.1.1;
    } ;
    group type external peeras 65502 gateway 10.0.0.1 {
        peer 10.0.0.1 ;
    } ;
    group type external peeras 65503 gateway 172.16.0.1 {
        peer 172.16.0.1 ;
    } ;
    group type internal peeras 64512 {
        peer 192.0.2.1
    } ;
} ;
[import and export statements omitted]

```

Example 2

Configure BGP-R in the `peer` statement.

```
autonomoussystem 64512 ;
routerid 192.0.2.1 ;
bgp on {
    group type external peeras 65501 {
        peer 192.168.1.1 gateway 192.168.1.1 ;
    } ;
    group type external peeras 65502 {
        peer 10.0.0.1 gateway 10.0.0.1 ;
    } ;
    group type external peeras 65503 {
        peer 172.16.0.1 gateway 172.16.0.1 ;
    } ;
    group type internal peeras 64512 {
        peer 192.0.2.1
    } ;
} ;
[import and export statements omitted]
```

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

"Third Party Route Advertisement" on page 70 of *Configuring GateD*

group type

Name

group type - specifies a BGP group as internal, external or routing

Syntax

```
group type external peeras autonomous_system
group type internal peeras autonomous_system
group type routing peeras autonomous_system proto protocol
```

Parameters

autonomous_system - a number between 1 and 65535, specifying a set of routers under a single technical administration and assigned by the Internet Assigned Numbers Authority

protocol - acronym of the protocol to be used to resolve BGP route next hops, including **static**, **rip**, **ospf**, **ospfase**, and **isis**. If **ospf** is specified, **ospfase** is automatically used as well. More than one protocol may be specified by using vertical bars to separate the acronyms. For example: **static | ospf**

Groups also have the following parameters, which are described throughout the BGP section of this manual:

```
ascount count
comm community_list
confed
gateway host
holdtime time
ignorefirstashop
interface interfacelist
keep ( all | none )
keepalivesalways
localtcp local_address
localas autonomous_system
logupdown
med
metricout metric
nexthopself
noaggregatorid
nogendefault
nov4asloop
outdelay time
passive
preference grouppreference
preference2 grouppreference2
recvbuffer buffer_size
reflector-client [ no-client-reflect ]
routetopeer
sendbuffer buffer_size
setpref metric
showwarnings
traceoptions trace_options
ttl ttl
```

Description

Within a group, BGP peers may be configured in one of two ways: They may be implicitly configured with the **allow** statement or explicitly configured with a **peer** statement. In **group type external**, full policy checking is applied to all incoming and outgoing advertisements. The external peers must be directly reachable through one of the machine's local interfaces. If they are not, BGP may be "multi-hopped" through the use of the **gateway** statement. The next hop transmitted is computed with respect to the shared interface.

The **group type internal** specifies an internal group operating where there is no IP-level IGP, for example, an SMDS network or MILNET. If the peer is not directly connected, the **route to peer** and **ttl** statements may be used to "multihop" the IBGP session.

Import and export policy may be applied to group advertisements. Routes received from external BGP peers are by default readvertised with the received metric when the **med** keyword is present.

The **group type routing** propagates routes between routers that are not directly connected. **group type routing** also computes immediate next hops for those external routes by using the BGP next hop that arrived with the route as a forwarding address to be resolved via an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, and the IGP is expected to carry only AS internal routes. The latter is used to find immediate next hops for the former. **proto protocol** names the interior protocol to be used to resolve BGP route next hops, and may be the name of any IGP in the configuration, including static. By default, the next hop in BGP routes advertised to **group type routing** peers will be set to the local address on the BGP connection to those peers because it is assumed that a route to this address will be propagated via the IGP. The **interface list** can optionally provide a list of interfaces whose routes are carried via the IGP for which third-party next hops can be used instead.

localtcp, **outdelay**, and **metricout** must be set in the **group** clause, not on a per-peer basis, for the group types **internal** and **routing**. If these options are set on the **peer** sub-clause, they must equal the values set on the corresponding **group** clause.

Default

none

Context

bgp

Examples

Example 1

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.2 ;  
        peer 192.0.2.3 ;  
    }  
}
```

```
    } ;
} ;
[import and export statements omitted]
```

Example 2

```
[autonomoussystem and routerid omitted]
bgp on {
    group type internal peeras 65534 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
} ;
[import and export statements omitted]
```

Example 3

```
[autonomoussystem and routerid omitted]
bgp on {
    group type routing peeras 64512 proto ospf {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
} ;
[import and export statements omitted]
```

Example 4

```
[autonomoussystem and routerid omitted]
bgp on {
    group type peeras 64512 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

holdtime

Name

holdtime - specifies the BGP hold time value, in seconds, to use when negotiating a peering session

Syntax

holdtime *time*

Parameters

time - time in seconds, an integer that is zero or at least 3

Description

holdtime specifies the BGP hold time value, in seconds, to use when negotiating a peering session with this group. If GateD does not receive a keepalive, update, or notification message within the period specified in the hold time field of the BGP Open message, then the BGP connection will be closed. The value must be at least 3.

The negotiated **holdtime** value is the lesser of the values sent in the exchanged BGP Open messages. If a *time* of zero is specified, no keepalives will be sent. If a *time* of zero is received from the remote peer, the **holdtime** must be configured to be zero in order for the peering session to become established.

Default

holdtime 180 ;

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

Example 1

holdtime set in the **group** statement

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 holdtime 30 {  
        peer 192.0.2.1 ; # Holdtime is 30 seconds  
        peer 192.0.2.2 ; # Holdtime is 30 seconds  
    } ;  
}
```

```
} ;
```

[import and export statements omitted]

Example 2

holdtime set in the group and peer statements

[autonomoussystem and routerid omitted]

```
bgp on {
```

```
    group type external peeras 64512 holdtime 90 {
```

```
        peer 192.0.2.1 ; # Holdtime is 90 seconds
```

```
        peer 192.0.2.2 holdtime 30 ; # Holdtime is 30 seconds
```

```
    } ;
```

```
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

keepalivesalways on page 258

ignorefirstashop

Name

`ignorefirstashop` - directs GateD to keep route server routes

Syntax

`ignorefirstashop`

Parameters

none

Description

Some routers, known as "route servers," are capable of propagating routes without appending their own autonomous system number to the AS Path. By default, GateD will drop such routes. Specifying `ignorefirstashop` on the `group` clause allows GateD to keep these routes. `ignorefirstashop` should be used only if there is no doubt that these peers are route servers and not normal routers.

Default

not enabled

Context

`bgp group type external` statement

`bgp peer` statement for external groups

Examples

Example 1

`ignorefirstashop` set in `group` statement

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 ignorefirstashop {  
        peer 192.0.2.1 ;  
        peer 192.0.2.2 ;  
    } ;  
} ;
```

[import and export statements omitted]

Example 2

```
ignorefirstashop set in peer statement  
[autonomoussystem and routerid omitted]  
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ignorefirstashop ;  
        peer 192.0.2.2 ;  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

ignore-nonprefixed-confederations

Name

ignore-nonprefixed-confederations - ignores malformed AS_PATHs containing non-prefixed confederation segments

Syntax

```
ignore-nonprefixed-confederations ;
```

Parameters

none

Description

ignore-nonprefixed-confederations causes BGP to ignore AS_PATHs that are received from BGP peers where there are BGP Confederation AS_PATH segments (AS_CONFED_SEQUENCE, AS_CONFED_SET) occurring anywhere other than at the left hand side of the AS_PATH. This feature is useful due to “buggy” routers on the Internet that will illegally advertise AS_PATHs containing confederation segments outside of a confederation boundary. If these routes propagate beyond the confederation boundary edge, they will cause the peering session of any router, which does not accept confederation segments from non-confederation peers, to drop the peering session and thus disrupt service. This option will cause routes containing non-prefixed confederation segments to be logged and stored in the RIB with a preference of -1. The GII command **show bgp bad-as-path** will then show all of these routes.

Default

not enabled

Context

bgp statement

Examples

```
routerid 192.0.0.1;  
autonomoussystem 64512;  
  
bgp on {  
    ignore-nonprefixed-confederations;  
  
    group type external peeras 65534 {  
        allow {  
            all;  
        };
```

```
};  
};  
[static, import, and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*
discard-nonprefixed-confederations on page 242
RFC 3065 - Autonomous System Confederations for BGP

keep

Name

keep - specifies whether to keep routes containing a router's own autonomous system (AS) number

Syntax

```
keep ( all | none )
```

Parameters

all or none

Description

keep all retains routes learned from a peer even if the routes' AS paths contain the router's own AS number. **keep none** causes GateD to discard routes containing the router's own AS number.

Default

```
keep none ;
```

Context

```
bgp group type external statement
```

```
bgp group type internal statement
```

```
bgp group type routing statement
```

```
bgp peer statement
```

Examples

Example 1

keep all set in **group** statement

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 keep all {  
        peer 192.0.2.1 ; # keep all  
        peer 192.0.2.2 ; # keep all  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

`keep all` set in `peer` statement

[`autonomous system` and `routerid` omitted]

```
bgp on {  
    group type external peer as 64512 keep none {  
        peer 192.0.2.1 keep all ;  
        peer 192.0.2.2 ; # keep none  
    } ;  
} ;
```

[`import` and `export` statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

keepalivesalways

Name

keepalivesalways - causes GateD to always send keepalives

Syntax

keepalivesalways

Parameters

none

Description

keepalivesalways causes GateD to always send keepalives, even when an update could have correctly substituted for one. **keepalivesalways** allows interoperability with routers that do not completely obey the protocol specifications on this point.

Default

disabled

Context

bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement

Examples

Example 1

keepalivesalways set in **group** statement
[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 keepalivesalways {  
        peer 192.0.2.1 ; # keepalivesalways  
        peer 192.0.2.2 ; # keepalivesalways  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

keepalivesalways set in peer statement

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 keepalivesalways ;  
        peer 192.0.2.2 ; # Normal behavior  
    } ;  
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

localas

Name

localas - identifies the autonomous system that GateD is representing to a group of peers

Syntax

localas *autonomous_system*

Parameters

autonomous_system - a number between 1 and 65535, specifying a set of routers under a single technical administration and assigned by the Internet Assigned Numbers Authority

Description

localas identifies the autonomous system that GateD is representing to a group of peers. The default is the globally configured **autonomoussystem** statement.

Default

inherited from the global **autonomoussystem**

Context

bgp group type external statement

Examples

```
[routerid omitted]
autonomoussystem 64512
bgp on {
    group type external peeras 65000 { # use as 64512 (inherited)
        peer 192.0.2.1 ; # keepalivesalways
    } ;
    group type external peeras 65001 localas 65534 {
        peer 192.0.2.2 ; # keepalivesalways
    } ;
} ;
[import and export statements omitted]
```

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

localtcp

Name

localtcp - specifies the address to be used on the local end of the TCP connection with a peer

Syntax

localtcp *local_address*

Parameters

local_address - 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 or in a dotted-quad format (xxx.xxx.xxx.xxx).

Description

localtcp specifies the IP address to be used on the local end of the TCP connection with the peer. For external peers, the local address must be on an interface that is shared with the peer or with the peer's gateway when **gateway** is used. A session with an external peer will be opened only when an interface with the appropriate local address (through which the peer or gateway address is directly reachable) is operating. For **internal** and **routing** peers, a peer session will be maintained when any interface with the specified local address is operating. In any case, an incoming connection will be recognized as a match for a configured peer only if it is addressed to the configured local address.

For group types **internal** and **routing**, set **localtcp** on the **group** clause.

Default

the IP address of a shared interface

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

[autonomous system and routerid omitted]

```
bgp on {  
    group type external peeras 64512 localtcp 192.168.1.1 {  
        peer 192.0.2.1 ;  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

logupdown

Name

logupdown - causes a message to be logged via the syslog mechanism whenever a BGP group enters or leaves the Established state

Syntax

logupdown

Parameters

none

Description

logupdown causes a message to be logged via the syslog mechanism whenever a BGP group enters or leaves the Established state

Default

disabled

Context

bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement

Examples

Example 1

Enables **logupdown** in the group

[**autonomoussystem** and **routerid** omitted]

```
bgp on {  
    group type external peeras 64512 logupdown {  
        peer 192.0.2.1 ; # logupdown enabled  
        peer 192.0.2.2 ; # logupdown enabled  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

Enable `logupdown` in the peer.

[`autonomous` system and `routerid` omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 logupdown ; # logupdown enabled  
        peer 192.0.2.2 ;           # logupdown disabled  
    } ;  
} ;
```

[`import` and `export` statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

`traceoptions` on page 295

med

Name

med - allows MEDs to be used in routing computations

Syntax

med

Parameters

none

Description

By default, any MED (Multi-Exit-Disc) received on a BGP connection is ignored. If MEDs are to be used in routing computations, the **med** option must be specified on the **group** or **peer** clauses. By default, MEDs are not sent on external connections. To send MEDs, use the metric option of the **export** statement or the **metricout** peer/group parameter.

When two routes to the same destination are received from different peers within the same peer-as, they could have different MEDs. When choosing between these routes, assuming that nothing else makes one preferable to the other (such as configured policy), the values of the differing MEDs are used to choose which route to use. In this comparison, the route with the lowest MED is preferred. Routes without MEDs are treated as having the best possible MED.

Default

disabled

Context

```
bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement
```

Examples

Example 1

Enables **med** processing on the **group** statement

[autonomoussystem and routerid omitted]

```
bgp on {
    group type external peeras 64512 med {
        peer 192.0.2.1 ; # med comparison is enabled
        peer 192.0.2.2 ; # med comparison is enabled
    }
}
```

```
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

Enables `med` processing on the `peer` statement

[`autonomous system` and `routerid` omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 med ; # med comparison is enabled  
        peer 192.0.2.2 ;    # med comparison is disabled  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

metricout

Name

metricout - causes BGP to send a Multi-Exit Discriminator (MED) when routes are advertised to peers

Syntax

metricout *metric*

Parameters

metric

Description

The **metricout** statement causes a BGP MED to be set on routes when they are advertised to peers. The metric hierarchy is as follows, starting from the most preferred:

1. the metric specified by export policy
2. peer-level **metricout**
3. group-level **metricout**
4. **defaultmetric**

For group types **internal** and **routing**, set **metricout** on the **group** clause instead of on the **peer** subclause. (MED needs to be common among all peers in an internal group or looping may occur.)

Default

no metric, or, if set, **defaultmetric**

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

Example 1

Set a **metricout** of 50 on the group.

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 metricout 50 {  
        peer 192.0.2.1 ; # Send MED of 50  
        peer 192.0.2.2 ; # Send MED of 50  
    }  
}
```

```
        peer 192.0.2.3 ; # Send MED of 50
    } ;
} ;
```

[import and export statements omitted]

Example 2

Set a `metricout` of 50 on a specific peer.

[`autonomoussystem` and `routerid` omitted]

```
bgp on {
    group type external peeras 64512 {
        peer 192.0.2.1 metricout 50 ; # Send MED of 50
        peer 192.0.2.2 ;                # Send no MED
        peer 192.0.2.3 ;                # Send no MED
    } ;
} ;
```

[import and export statements omitted]

Example 3

Set a `metricout` of 50 on a specific peer.

[`autonomoussystem` and `routerid` omitted]

`defaultmetric 100;`

```
bgp on {
    group type external peeras 64512 {
        peer 192.0.2.1 metricout 50 ; # Send MED of 50
        peer 192.0.2.2 ;                # Send MED of 100
        peer 192.0.2.3 ;                # Send MED of 100
    } ;
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

`defaultmetric` on page 241

nexthopself

Name

nexthopself - sets this group or peer's next hop to the router's own address on advertisement

Syntax

nexthopself

Parameters

none

Description

nexthopself sets this group's next hop to the router's own address even if it would normally be possible to send a third-party next hop.

nexthopself can cause inefficient routes to be followed. It might be needed in some cases to deal with improperly bridged interconnect media (in cases where the routers on the "shared" medium do not really have full connectivity to each other), or when political situations cause broken links.

nexthopself can be used only for external groups.

Default

disabled

Context

bgp group type external statement

bgp peer statement (in external groups)

Examples

Consider the following topology:

Three routers that have interfaces all in the same subnet are in a partially meshed ATM network.

```
R1 -- R2
    |
    R3
```

- R1 and R2 have a Permanent Virtual Circuit (PVC.)
- R2 and R3 have a PVC.
- R1 and R3 do not have any direct connectivity.
- R1, R2, and R3 share a common subnet.

Because all routers are in the same subnet, normally, routes exchanged between R1, R2, and R3 would all use third-party routing. Routes exchanged between R1 and R3 would result in a blackhole because R1 and R3 do not have a valid physical connection.

When R2 exchanges routes with R1 and R3, it should use `nexthopself` to disable the third-party routing.

Configuration for R2:

[`autonomous-system` and `router-id` omitted]

```
bgp on {  
    group type external peeras 64512 nexthopself {  
        peer 192.0.2.1 ; # R1  
    } ;  
    group type external peeras 64513 nexthopself {  
        peer 192.0.2.2 ; # R2  
    } ;  
} ;
```

[`import` and `export` statements omitted]

This could alternatively be written as:

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 nexthopself ; # R1  
    } ;  
    group type external peeras 64513 {  
        peer 192.0.2.2 nexthopself ; # R2  
    } ;  
} ;
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Third Party Route Advertisement” on page 70 of *Configuring GateD*

noaggregatorid

Name

`noaggregatorid` - causes GateD to specify the `routerid` in the aggregator attribute as 0

Syntax

`noaggregatorid`

Parameters

none

Description

`noaggregatorid` causes GateD to specify the `routerid` in the aggregator attribute as 0 (instead of the `routerid` of the router) in order to prevent different routers in an AS from creating aggregate routes with different AS paths.

Default

disabled

Context

`bgp group type external statement`
`bgp group type internal statement`
`bgp group type routing statement`
`bgp peer statement`

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    group type external peeras 64512 noaggregatorid {
        peer 192.0.2.1 ;
    } ;
# Any aggregates will have a routerid of 0.0.0.0
    aggregate 10.0.0.0 masklen 9 {
        proto bgp {
            10.0.2.0 masklen 24 ;
            10.0.4.0 masklen 24 ;
        } ;
    } ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Chapter 33 Route Aggregation and Generation” on page 155 of *Configuring GateD*

nogendefault

Name

nogendefault - prevents GateD from generating a default route when BGP receives a valid update from its peer

Syntax

nogendefault

Parameters

none

Description

nogendefault prevents GateD from generating a default route when BGP receives a valid update from its peer. The default route is generated only when the **gendefault** option is enabled.

Default

disabled

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 nogendefault {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

"Chapter 6 Options Statements" on page 21 of *Configuring GateD*

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

no-mp-nexthop

Name

no-mp-nexthop - specifies to not send a standard BGP nexthop when only sending multi-protocol BGP routes

Syntax

no-mp-nexthop

Parameters

none

Description

According to the BGP multi-protocol specification, RFC 2858, when a BGP Speaker is only sending multi-protocol routes and is not sending any reachability in the standard BGP NLRI field, it is not necessary to send the standard BGP NEXT_HOP path attribute. Unfortunately, some well-deployed implementations have a bug where this field must be present even if it is ignored.

By default, GateD will send the standard NEXT_HOP field, even when only multi-protocol routes are being sent. If the reachability is IPv4, the NEXT_HOP field will be the same as the multi-protocol nexthop. In the case of non-IPv4 reachability, the standard NEXT_HOP field will contain "0.0.0.0".

Defaults

not set

Context

```
bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement
```

Examples

```
bgp on {
    group type external peeras 65412 no-mp-nexthop {
        peer 192.0.2.1 # no-mp-nexthop enabled
    } ;
    group type internal peeras 65534 {
        peer 192.168.1.1 ; # no-mp-nexthop disabled
        peer 19.168.2.1 no-mp-nexthop ; # no-mp-nexthop enabled
    } ;
}
```

} ;

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

nov4asloop

Name

nov4asloop - prevents routes with looped AS paths from being advertised to version 4 external peers

Note: **nov4asloop** is deprecated.

Syntax

nov4asloop

Parameters

none

Description

nov4asloop prevents routes with looped AS paths from being advertised to version 4 external peers. Use **nov4asloop** to avoid advertising routes to peers that would incorrectly forward the routes on to version 3 peers. In this context, “looped” refers to “AS Path stuffing” where a given AS is inserted multiple times in an AS Path. (**nov4asloop** is superseded by **ascount**.)

Default

disabled

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 nov4asloop {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Chapter 30 BGP Communities” on page 133 of *Configuring GateD*

open-on-accept

Name

open-on-accept - causes GateD to send a BGP Open message immediately after the transport (TCP) connection has completed

Syntax

open-on-accept

Parameters

None

Description

When GateD receives an incoming BGP peering session, it will delay sending the BGP Open message for a short time after the transport connection (TCP) has completed. This is a violation of the BGP Finite State Machine, but is used to allow the calling peer to send its Open message first. This is used to determine if a peering session matches a given **allow** clause. When the **open-on-accept** keyword is present, GateD will immediately send the Open message when the TCP connection has completed for configured peers. If the peer is unconfigured (is matched by an **allow** clause, but not a **peer** keyword), or is **passive**, GateD will continue to wait for the Open message from the remote peer before sending its own BGP Open message.

Default

Off

Context

bgp statement

Examples

```
autonomoussystem 64512;  
bgp yes {  
    open-on-accept ;  
    group type external peeras 65534 {  
        peer 192.168.1.1;  
    };  
};
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

outdelay

Name

outdelay - damps route fluctuations

Syntax

outdelay *time*

Parameters

time - time in seconds

Description

outdelay damps route fluctuations. The **outdelay** time is the amount of time a route must be present in the GateD routing database before it is exported to BGP.

Note: Weighted Route Damping may be better suited for improving overall network stability. The use of this option may delay route convergence for well-behaved routers.

Default

outdelay 0 ; (This feature is disabled.)

Context

bgp group type external statement

bgp peer statement for external groups

Examples

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 outdelay 10 {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

passive

Name

passive - prevents GateD from ever trying to open a BGP connection with peers in this group

Syntax

passive

Parameters

none

Description

passive prevents GateD from ever trying to open a BGP connection with peers in this group. Instead, the router will wait for the peer to initiate a connection. **passive** was introduced to handle a problem in BGP3 and earlier in which two peers might both attempt to initiate a connection at the same time. This problem is fixed in the BGP4 protocol, so the passive option is not needed with BGP4 sessions.

Note: If it is applied to both sides of a peering session, **passive** will prevent the session from ever being established. For this reason, and because it is generally not needed, the use of **passive** is discouraged.

Default

disabled

Context

bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    group type external peeras 64512 passive {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

peer

Name

peer - configures an individual peer

Syntax

```
peer host [ peer options ] ;
```

Parameters

host - the IP address of the host machine, in dotted-quad format (xxx.xxx.xxx.xxx)

Peer options are described throughout the BGP section and include:

```
ascount count
gateway gateway
ignorefirstashop
holdtime time
keep ( all | none )
keepalivesalways
localtcp local_address
logupdown
med
metricout metric
noagggregatorid
nogendefault
nov4asloop
outdelay
passive
preference peerpreference
preference2 peerpreference2
recvbuffer buffersize
routetopeer
sendbuffer buffersize
showwarnings
traceoptions trace_options
ttl ttl
```

Description

peer configures an individual peer. Each peer inherits all parameters specified on a **group** clause as defaults. Many defaults may be overridden by parameters explicitly specified on the **peer** subclause. Within each **group** clause, individual peers can be specified, or a group of potential peers can be specified using **allow**. Use **allow** to specify a set of address masks. If GateD receives a BGP connection request from any address in the set specified, it will accept it and set up a peer relationship.

Default

not applicable

Context

`bgp` statement

Examples

[`autonomous` `system` and `routerid` omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ;  
        peer 10.0.0.1 passive ;  
        peer 192.168.1.1 nexthopself ;  
    } ;  
} ;
```

[`import` and `export` statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

preference

Name

preference - specifies how active routes that are learned from BGP (compared to other protocols) will be selected

Syntax

preference *bgppreference*

Parameters

bgppreference - an assigned integer between 0 (directly connected) and 255 (for EGP)

Description

preference specifies how active routes that are learned from BGP (compared to other protocols) 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. This **preference** may be overridden by a **preference** specified on the **group** or **peer** statements, or by import policy.

Default

preference 170 ;

Context

bgp statement

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

Example 1

Set the global BGP preference to 140, which makes BGP routes better than OSPF AS-external routes.

[autonomoussystem and routerid omitted]

```
bgp on {  
    preference 140 ;  
    group type external peeras 64512 {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

Example 2

Set the preference for a specific BGP group to be better than routes from another BGP group. This ensures that, for a given prefix, these routes are always preferred.

[autonomoussystem and routerid omitted]

```
bgp on {  
  # These routes are always preferred compared with  
  # other BGP routes.  
    group type external peeras 64512 preference 165 {  
      peer 192.0.2.1 ;  
    } ;  
    group type external peeras 65000 { # default preference of 170  
      peer 192.0.2.2 ;  
    } ;  
} ;
```

[import and export statements omitted]

Example 3

Policy over all.

[autonomoussystem and routerid omitted]

```
bgp on {  
  group type external peeras 64512 {  
    peer 192.0.2.1 ;  
  } ;  
  import proto bgp as 64512 {  
    # We will prefer this route over almost anything other than  
    # a directly attached interface.  
    10.0.0.0 masklen 24 preference 5 ;  
    all ; # default 170  
  } ;  
} ;
```

[export statement omitted]

See Also

“Chapter 3 Preferences and Route Selection” on page 11 of *Configuring GateD*

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

preference2

Name

preference2 - breaks a **preference** tie between groups

Syntax

preference2 *group***preference2**

Parameters

*group***preference2** - an integer between 0 and 255

Description

preference2 breaks a **preference** tie between groups. Preferences are the first criteria of comparison for route selection.

Default

preference2 0 ;

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

[autonomoussystem and routerid omitted]

bgp on {

Prefer these routes

group type external peeras 64512 preference 5 {

peer 192.0.2.1 ;

} ;

group type external peeras 65000 { # default preference 170

Routes from this peer are preferred over 192.0.2.3

peer 192.0.2.2 preference2 10 ;

peer 192.0.2.3 preference2 20 ;

} ;

} ;

[import and export statements omitted]

See Also

“Chapter 3 Preferences and Route Selection” on page 11 of *Configuring GateD*

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

recvbuffer

Name

recvbuffer - controls the amount of memory requested from the kernel for the receive buffer

Syntax

recvbuffer *buffer_size*

Parameters

buffer_size - an integer between 1 and 65535

Description

recvbuffer controls the amount of memory requested from the kernel for the receive buffer. The maximum supported is 65535 bytes, although many kernels have a lower limit. By default, GateD configures the maximum supported. **recvbuffer** is not needed on normally functioning systems.

Default

recvbuffer 65535 ;

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 recvbuffer 32768 {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

"Chapter 14 Border Gateway Protocol (BGP)" on page 61 of *Configuring GateD*

sendbuffer on page 291

reflector-client [no-client-reflect]

Name

reflector-client - specifies that GateD will act as a route reflector for this group
no-client-reflect - specifies that GateD will not act as an intra-group reflector and, thus, will not reflect routes back to peers within the same group. If used, this keyword must follow **reflector-client**.

Syntax

```
reflector-client [ no-client-reflect ]
```

Parameters

none

Description

reflector-client specifies that GateD will act as a route reflector for this group.
no-client-reflect specifies that GateD will not act as an intra-group reflector and thus will not reflect routes back to peers within the same group. This is used when client peers within a route-reflection group are fully meshed.

Default

none

Context

bgp group type internal statement
bgp group type routing statement

Examples

```
[autonomoussystem omitted]
routerid 192.0.2.1 ;
bgp on {
    clusterid 192.0.2.254 ;
    group type internal peeras 65534 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
    # Routes received from clients will have the clusterid of
    # 192.0.2.254 added to their BGP attributes
    group type internal peeras 65534 reflector-client {
        peer 192.0.2.10 ;
        peer 192.0.2.11 ;
        peer 192.0.2.12 ;
        peer 192.0.2.13 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

“Route Reflection” on page 71 *Configuring GateD*

routepeer

Name

routepeer - specifies the actual time to live (TTL) used on a socket in all cases

Syntax

routepeer

Parameters

none

Description

routepeer specifies the actual TTL used on a socket in all cases. In particular, if GateD realizes that two BGP speakers are peering over a single network, GateD automatically sets the **dontroute** option on their peering session. This, in turn, causes the TTL of the packets to be set to 1. **routepeer** prevents the **dontroute** option from being set. If you specify **routepeer**, but don't specify a TTL, and you are directly connected, GateD will set the TTL of your peering session to 1. If you want a TTL greater than 1 for directly connected peers, you must specify both **routepeer** and the **ttl** that you require.

Default

disabled

Context

bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    group type external peeras 64512 {
        peer 192.0.2.1 routepeer ttl 5 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*
ttl on page 297

sendbuffer

Name

sendbuffer - controls the amount of send buffering asked of the kernel

Syntax

sendbuffer *buffer_size*

Parameters

buffer_size - an integer between 1 and 65535

Description

sendbuffer controls the amount of memory requested from the kernel for the send buffer. The maximum supported is 65535 bytes, although many kernels have a lower limit. By default, GateD configures the maximum supported. **sendbuffer** is not needed on normally functioning systems.

Default

sendbuffer 65535 ;

Context

bgp group type external statement

bgp group type internal statement

bgp group type routing statement

bgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    group type external sendbuffer 32768 {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

recvbuffer on page 287

setpref

Name

setpref - allows BGP's Local_Pref attribute to be used to set the GateD preference on reception, and allows GateD preference to set the Local_Pref on transmission

Syntax

setpref *metric*

Parameters

metric

Description

setpref allows BGP's Local_Pref attribute to be used to set the GateD preference on reception, and allows GateD preference to set the Local_Pref on transmission. The **set-pref** metric works as a lower limit, below which the imported Local_Pref may not set the GateD preference. (For full details, see "Preferences and Route Selection" on page 11 in *Configuring GateD*.)

Default

none

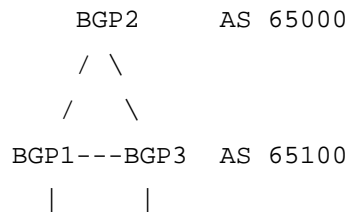
Context

bgp group type internal statement

bgp group type routing statement

Examples

For this topology,



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

Example 1

BGP1 configuration:

```
bgp on {
    group type external peeras 65000 {
        peer 10.0.0.2; # BGP2
    };
};
```

```
    group type internal peeras 65100 setpref 100 {  
        peer 192.168.10.2; # BGP3  
    };  
};
```

Example 2

BGP3 configuration:

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

See Also

“Chapter 3 Preferences and Route Selection” on page 11 in *Configuring GateD*

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

showwarnings

Name

showwarnings - causes GateD to issue warning messages when receiving questionable BGP updates such as duplicate routes and/or deletions of non-existing routes

Syntax

showwarnings

Parameters

none

Description

showwarnings causes GateD to issue warning messages when receiving questionable BGP updates such as duplicate routes and/or deletions of non-existing routes. Normally, these events are silently ignored.

Default

disabled

Context

bgp group type external statement
bgp group type internal statement
bgp group type routing statement
bgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    group type external peeras 64512 showwarnings {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*
traceoptions on page 295

traceoptions

Name

traceoptions - specifies the tracing options for this group

Syntax

```
traceoptions trace_options
```

Parameters

trace options include:

- packets** - Trace all BGP packets.
- open** - Trace BGP Open packets.
- update** - Trace BGP Update packet.
- keepalive** - Trace BGP Keepalive packets.
- all** - Trace changes to the GateD routing table.

Description

traceoptions specifies the tracing options for this group. By default, these are inherited from the BGP or global trace options. These values may be overridden on the **peer** statements.

Default

inherited from global **traceoptions**

Context

- bgp** statement
- bgp group** statement
- bgp peer** statement

Examples

```
[autonomoussystem and routerid omitted]
bgp on {
    traceoptions packets ;
    group type external peeras 64512 {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

showwarnings on page 294

ttl

Name

`ttl` - specifies time to live

Syntax

`ttl ttl`

Parameters

`ttl` - ttl has two units: seconds and number of hops. Either can be used.

Description

The `ttl` option is provided mainly when attempting to communicate with improperly functioning routers that ignore packets sent with a `ttl` of 1. Not all kernels allow the TTL to be specified for TCP connections.

Default

By default, GateD sets the IP TTL for local peers to 1, and the TTL for non-local peers to the default kernel value.

Context

`bgp group type routing` statement

`bgp peer` statement

Examples

[autonomoussystem and routerid omitted]

```
bgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ; # ttl of 1 (directly connected network)  
        peer 192.168.1.1 routetopeer ttl 2 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

“Chapter 14 Border Gateway Protocol (BGP)” on page 61 of *Configuring GateD*

`routetopeer` on page 290

Chapter 11

Router Discovery

address

Name

address - specifies parameters for logical addresses

Syntax

```
address interface_list
    [ advertise | ignore ]
    [ broadcast | multicast ]
    [ ineligible | preference preference ]
;
```

Note: At least one option for **address** must be specified for the command to parse.

Parameters

interface_list - specifies the set of addresses to which the other parameters apply

advertise - specifies that the addresses in the *interface_list* should be included in router advertisements

ignore - specifies that the addresses in the *interface_list* should not be included in router advertisements

broadcast - specifies that the addresses in the *interface_list* should be included in broadcast router advertisements

multicast - specifies that the addresses in the *interface_list* should be included in multicast router advertisements

preference *preference* - specifies that the addresses in the *interface_list* should be advertised with a preference of *preference*. Preference is a 32-bit, signed, two's complement integer, with higher values being more preferable.

ineligible - specifies that the addresses in the *interface_list* should be advertised with the worst possible preference, making them ineligible to become the default routers

Description

The **address** statement is used to specify the set of addresses that should be advertised, and the parameters with which they should be specified.

If an interface does not support multicast, but multicast is specified for that interface in the **routerdiscovery** clause, then that interface will not be advertised at all.

Default

```
address interface_list
    advertise
    multicast
    preference 0
;
```

Context

routerdiscovery server statement

Examples

```
routerdiscovery server on {
    address 223.1.25.3 ignore ;
    address 221.3.5.8 preference 10 ;
} ;
```

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*

routerdiscovery server on page 308

interface (client)

Name

interface - specifies the physical interfaces on which the router discovery client is to run

Syntax

```
interface phys_interface_list
    [ enable | disable ]
    [ multicast | broadcast ]
    [ quiet | solicit ]
;
```

Parameters

phys_interface_list - specifies the set of physical interfaces to which the rest of the parameters apply

enable - specifies that router discovery should be performed on the specified interfaces

disable - specifies that router discovery should not be performed on the specified interfaces

multicast - specifies that router discovery should be performed on the specified interfaces, unless disabled, and that router solicitations should be multicast rather than broadcast

quiet - specifies that no router solicitations should be sent on these interfaces even though router discovery will still be performed

solicit - specifies that initial router solicitations will be sent on these interfaces

Description

The **interface** statement specifies the set of interfaces on which the router discovery client should be run. It further allows the specification of how the protocol should be run on the given interfaces. If no interfaces are cited in the **routerdiscovery client** clause, the router discovery client will be **on** for all interfaces. If, however, any interfaces are cited within the **routerdiscovery client** clause, router discovery will be **on** for only those interfaces.

Defaults

```
interface all enable
    enable
    multicast
;
```

Context

routerdiscovery client statement

Examples

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

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

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*
`routerdiscovery client` on page 306

interface (server)

Name

interface - specifies the physical interfaces on which to run the server

Syntax

```
interface phys_interface_list
    [ maxadvinterval max_time ]
    [ minadvinterval min_time ]
    [ lifetime life_time ]
;
```

Parameters

phys_interface_list - specifies a list of physical interfaces to run the router discovery server on. **all** can be used to specify all interfaces.

maxadvinterval *max_time* - specifies the maximum time allowed between sending broadcast or multicast router advertisements from the interface. This must be no less than 4 seconds and no more than 30 minutes. The default is 10 minutes.

minadvinterval *min_time* - specifies the minimum time allowed between sending unsolicited broadcast or multicast router advertisements from the interface. This must be no less than 3 seconds and no more than **maxadvinterval**. If **minadvinterval** is set to greater than **maxadvinterval**, GateD will reconfigure the value of **minadvinterval** to be equal to the value of **maxadvinterval** and log the change. The default is $0.75 * \text{maxadvinterval}$.

lifetime *life_time* - specifies how long the addresses in a given router advertisement are valid. Lifetime must be no less than **maxadvinterval** and no more than 2 hours and 30 minutes. If **lifetime** is set to less than **maxadvinterval**, GateD will reconfigure **lifetime** to be equal to **maxadvinterval**. The default is $3 * \text{maxadvinterval}$.

Description

The **interface** statement specifies the set of physical interfaces on which the router discovery server is to run. It also allows specification of various timers associated with the physical interfaces. Note a slight difference in convention from the rest of GateD: **interface** specifies a list of physical interfaces (such as **le1**, **ef0**, and **en1**), while **address** specifies a list of IP addresses.

If no interfaces are cited in the **routerdiscovery server** clause, the router discovery server will be **on** for all interfaces. If, however, any interfaces are cited within the **routerdiscovery server** clause, routerdiscovery will be **on** for only those interfaces.

Default

```
interface all
    ( maxadvinterval 00:10:00
      minadvinterval .75*max_time
```

```
lifetime 3*max_time )  
;
```

Context

`routerdiscovery server` statement

Examples

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;  
}
```

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*

`routerdiscovery server` on page 308

preference

Name

preference - specifies the preferability of the address as a default router address, relative to other router addresses on the same subnet

Syntax

preference *preference*

Parameters

preference - the *preference* value to be used for routes learned from the router discovery protocol

Description

The **preference** command specifies the preference value to be used when comparing routes learned via the router discovery protocol against routes learned via other protocols. Higher values are more preferred.

Defaults

preference 0

Context

routerdiscovery client statement

Examples

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

```
routerdiscovery client on {  
    preference 3 ;  
    interface fxp0 enable multicast solicit ;  
} ;
```

The following example demonstrates preference as applied to a **routerdiscovery** server:

```
routerdiscovery server on {  
    address 1.2.3.4  
    preference 50 ;  
} ;
```

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*

routerdiscovery client on page 306

routerdiscovery client

Name

routerdiscovery client - enables or disables the client portion of the router discovery protocol

Syntax

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

Parameters

on | off - enables or disables the client portion of the router discovery protocol

traceoptions - specifies which information to log for router discovery

preference - specifies how router discovery default routes are compared to default routes from other protocols

interface - specifies the physical interfaces on which the router discovery client is to run

Description

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.

Default

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

Context

global statement

Examples

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

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

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*

routerdiscovery server

Name

routerdiscovery server - enables or disables the server portion of the router discovery protocol

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 ]  
    ;  
} ] ;
```

Parameters

on | off - enables or disables the server portion of the router discovery protocol

traceoptions - specifies which information to log for router discovery

address - specifies parameters for logical addresses

interface - specifies the physical interfaces on which the router discovery server is to run

Description

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.

The router discovery server runs on routers and announces their existence to hosts. The router discovery server announces hosts' existence by periodically multicasting or broadcasting a router advertisement to each interface on which it is enabled. These router advertisements contain a list of all the routers' addresses on a given interface and the preference of each address for use as the default router on that interface.

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 is 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 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.

Default

```
routerdiscovery server off [ {
    interface all
        ( maxadvinterval 00:10:00
          minadvinterval .75*max_time
          lifetime 3*max_time )
    ;
    address interface_list
        advertise
        multicast
        preference 0
    ;
} 1 ;
```

Context

global statement

Examples

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;  
}
```

See Also

“Chapter 15 Router Discovery” on page 85 of *Configuring GateD*

`routerdiscovery client` on page 306

traceoptions

Name

traceoptions - specifies which information to log for router discovery

Syntax

```
traceoptions traceoptions ;
```

Parameters

traceoptions

Description

traceoptions specifies the router discovery tracing options. The Router Discovery Client and Server support the **state** trace flag, which traces various protocol occurrences.

The Router Discovery Client and Server do not directly support any packet tracing options. Tracing of router discovery packets is enabled via the ICMP Statement. See "Chapter 16 Internet Control Message Protocol (ICMP)" on page 89 in *Configuring GateD* for more information on ICMP.

Defaults

defaults to global traceoptions

Context

routerdiscovery server statement

routerdiscovery client statement

Examples

The following example turns on all traceoptions for the router discovery server, and logs the information in a file called "rdisc" in the directory /var/tmp.

```
routerdiscovery server on {  
    traceoptions "/var/tmp/rdisc" all;  
};
```

The following example turns on all traceoptions for the router discovery client, and logs the information in a file called "rdisc" in the directory /var/tmp.

```
routerdiscovery client on {  
    traceoptions "/var/tmp/rdisc" all;  
};
```

See Also

"Chapter 15 Router Discovery" on page 85 of *Configuring GateD*

"Chapter 4 Trace Statements" on page 15 of *Configuring GateD*

`routerdiscovery client` on page 306

Chapter 12

Internet Control Message Protocol (ICMP) Statement

icmp

Name

icmp - this clause allows for commands pertaining to the Internet Control Message Protocol (ICMP). Currently, only **traceoptions** is supported.

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 ) ];  
};
```

Parameters

traceoptions - sets tracing options for ICMP interactions

Description

On systems without the BSD routing socket, GateD listens to 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 in *Configuring GateD* 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 is allowed by any GateD **traceoptions** clause. This allows for easy separation of non-redirect ICMP messages from redirect messages in the trace file.

Default

no ICMP tracing

Context

global statement

Examples

See `icmp traceoptions`.

See Also

"Chapter 16 Internet Control Message Protocol (ICMP)" on page 89 of *Configuring GateD*
`traceoptions` on page 3

traceoptions

Name

traceoptions - sets the logging options for ICMP interactions

Syntax

```
traceoptions none ;
traceoptions
    [ tracefile [ replace ] [ size tracesize [ k | m ] files tracefiles ]
]
    [ nostamp ] [ trace_global_options | trace_protocol_packets ]
    [ except ( trace_global_options | trace_protocol_packets ) ] ;
```

Parameters

none - Don't trace any ICMP packets.

replace - specifies to start tracing by truncating an existing file. The default is to append to an existing file.

size tracesize [k | m] files tracefiles - Limits the maximum size of the trace file to the specified *tracesize* (minimum 10k). When the trace file reaches the specified size, the file is renamed to file.0, then file.1, then file.2, up to the maximum number of files (the minimum specification is 2).

k - specifies the file size in kilobytes

m - specifies the file size in megabytes

files tracefiles - specifies the maximum number of files

nostamp - specifies that a timestamp should not be prepended to all trace lines

except - the options listed below are not traced.

trace_global_options - global trace options as defined in "Chapter 4 Trace Statements" on page 15 in *Configuring GateD*.

trace_protocol_packets - [detail] [send | receive] (packets | redirect | routerdiscovery | info | error) as defined below:

detail - Use detailed packet tracing.

send - Trace only ICMP packets sent.

receive - Trace only ICMP packets received.

packets - Trace all ICMP packet types.

redirect - Trace only ICMP redirect packets.

routerdiscovery - Trace only ICMP router discovery packets.

info - Trace only ICMP informational packets, which include:

- mask request/response
- info request/response
- echo request/response
- time stamp request/response

error - Trace only ICMP error packets, which include:

- time exceeded
- parameter problem
- unreachable
- source quench

Description

The ICMP command allows control over which packets are traced and logged. It inherits the defaults from the global `traceoptions` command (see `traceoptions`) and overrides them for ICMP. Tracing is controlled on a packet level, or turned off with `none`. The `detail` tag specifies that more detailed information should be logged. Only packets sent or received can be traced, and tracing can be controlled for various types of packets. Finally, types can be combined to get sets, and "`except`" can be used to reject specific tracing.

Default

no ICMP tracing

Context

`icmp` statement

Examples

Example 1

Turn off ICMP tracing.

```
icmp {  
    traceoptions none;  
} ;
```

Example 2

Trace all ICMP packets with detail.

```
icmp {  
    traceoptions detail packets;  
} ;
```

Example 3

Trace only redirect packets sent, with `detail`.

```
icmp {  
    traceoptions detail send redirect;  
} ;
```

Example 4

Trace all ICMP packets except router discovery to separate files with a size of 1024K and 3 total files, replacing them as necessary.

```
icmp {  
    traceoptions "/var/tmp/icmp.log" replace size 1024k files 3  
    detail packets except routerdiscovery;  
} ;
```

Example 5

Trace, in detail, received info and error ICMP packets.

```
icmp {  
    traceoptions detail recv info error;  
} ;
```

Example 6

This example shows another way of tracing, in detail, all sent and received packets.

```
icmp {  
    traceoptions detail redirect routerdiscovery info error;  
} ;
```

See Also

“Chapter 16 Internet Control Message Protocol (ICMP)” on page 89 of *Configuring GateD*

`traceoptions` on page 3

Chapter 13

Redirect Processing

interface

Name

interface - allows the enabling and disabling of redirects on an interface basis

Syntax

```
interface interface_list [noredirects] [redirects];
```

Parameters

interface_list - specifies the set of interfaces to which the rest of the parameters apply, or **all** for all interfaces

noredirects - specifies that redirects received via the specified interfaces will be ignored

redirects - specifies that redirects received via the specified interface will be accepted

Description

interface specifies the set of interfaces on which the redirects are enabled or disabled.

Defaults

interface_list - Redirects are processed from all interfaces unless an interface statement is configured, in which case only redirects on the specified interfaces are processed. Interfaces may be specified by name or address.

Context

redirect statement

Examples

The following example enables redirects on interface fxp0:

```
redirect on {  
    interface fxp0 redirects;  
} ;
```

See Also

`redirect` on page 322

preference

Name

preference - sets the preference for a route learned from a redirect

Syntax

preference *preference* ;

Parameters

preference - the value to be used for routes learned from the redirect protocol

Description

preference specifies the preference value to be used when comparing routes learned via the redirect protocol against routes learned via other protocols.

Defaults

The default value of *preference* is 30.

Context

redirect statement

Examples

The following example sets the preference for a route learned from a redirect to be 45.

```
redirect on {  
    preference 45;  
} ;
```

See Also

redirect on page 322

redirect

Name

redirect - specifies how GateD manages ICMP redirects

Syntax

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

Parameters

preference *preference* - the value to be used for routes learned from the redirect protocol

interface *interface_list* - specifies the set of interfaces to which the rest of the parameters apply, or **all** for all interfaces

noredirects - specifies that redirects received via the specified interfaces will be ignored

redirects - specifies that redirects received via the specified interface will be accepted

trustedgateways *gateway_list* - a list of host names or addresses

traceoptions *trace_options* - redirect tracing options

Description

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.

Defaults

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

Context

global

Examples

```
redirect on {
```

```
    traceoptions "/tmp/redirlog" all;  
    interface all noredirects;  
};
```

See Also

“Chapter 16, Redirect Processing” on page 91 in *Configuring GateD*

traceoptions

Name

traceoptions - specifies which information to log for redirect

Syntax

```
traceoptions trace_options ;
```

Parameters

trace_options

Description

traceoptions specifies the redirect tracing options. There are no redirect-specific tracing options. All non-error messages are traced when normal is specified for *trace_options*.

Defaults

global traceoptions

Context

redirect statement

Examples

The following example turns on all traceoptions for the redirect protocol and logs the information in a file called "redirect" in the directory /var/tmp.

```
redirect on {  
    traceoptions "/var/tmp/redirect" all;  
};
```

See Also

"Trace Statements" on page 15 in *Configuring GateD*

trustedgateways

Name

trustedgateways - specifies that redirects will be accepted only from the *gateway_list*

Syntax

```
trustedgateways gateway_list ;
```

Parameters

gateway_list - a list of host names or addresses

Description

trustedgateways defines the list of gateways from which redirects will be accepted where the *gateway_list* is a list of host names or addresses.

Defaults

By default, all routers on the shared network(s) are trusted to supply redirects. But if **trustedgateways** is specified, only redirects from the *gateway_list* in the list are accepted.

Context

redirect statement

Examples

In the following example, only redirects received via interface fxp0 and from a host with an ip address of 10.0.0.0 will be accepted.

```
redirect on {  
    interface fxp0 redirects;  
    trustedgateways 10.0.0.0;  
} ;
```

See Also

redirect on page 322

Chapter 14

Kernel Interface

background

Name

background - controls the background processing of routes

Syntax

```
background [ limit number ] [ priority ( flash | higher | lower ) ] ;
```

Parameters

limit *number* - **limit** specifies the number of routes that may be processed during one batch. *number* must be in the range 0 to 4,294,967,295. Note that *number* is an unsigned integer. Specifying -1 results in the maximum 32-bit unsigned number, which is mentioned above.

priority (flash | higher | lower)

priority specifies the priority of the processing of batches of kernel updates in relationship to the flash update processing. The default is **lower**, which means that flash updates are processed first. To process kernel updates at the same priority as flash updates, specify **flash**. To process kernel updates at a higher priority, use **higher**.

Description

Because only interface routes are normally installed during a flash update, the remaining routes are processed in batches in the background, that is, when no routing protocol traffic is being received.

Defaults

Process up to 120 routes at a time, processing flash updates first.

```
kernel { background limit 120 priority lower ; } ;
```

Context

kernel statement

Examples

Example 1

Process 100 routes at a time.

```
kernel { background limit 100; } ;
```

Example 2

Process kernel updates first.

```
kernel { background priority higher ; } ;
```

See Also

“Chapter 18 Kernel Interface” on page 95 of *Configuring GateD*

flash on page 329

kernel on page 327

flash

Name

flash - used to control the number and type of routes processed during a flash update

Syntax

```
flash [ limit number ] [ type ( interface | interior | all ) ] ;
```

Parameters

limit *number* - **limit** specifies the maximum *number* of routes that may be processed during one **flash** update. The default is 20. A value of -1 will cause all pending route changes of the specified type to be processed during the flash update. *number* is essentially unlimited. It must be in the range 0 to 4,294,267,295. Note that *number* is unsigned. Specifying **flash limit -1 all** causes all routes to be installed during the flash update; this mimics the behavior of prior versions of GateD that did not support this kernel option.

type (interface | interior | all) - **type** specifies the type of routes that will be processed during a **flash** update (see "Description" below). **interior** specifies that interior routes will be installed. **all** specifies the inclusion of exterior routes as well. The default is **interface**, which specifies that only interface routes will be installed during a flash update.

Description

When routes change as a result of kernel or protocol module activity, the process of notifying the GateD protocol module is called a flash update. The kernel forwarding table interface is the first to be notified. The flash process is concerned with three types of routes:

- interface routes: routes defined by an interface statement
- interior routes: routes within the domain
- exterior routes: routes exterior to the domain

A flash update results from protocol activity. The intent of this mechanism is to limit the number of routes installed during a flash update -- suspending the current protocol module until the flash completes. Typically, only up to 20 interface routes are normally installed during a flash update. The remaining routes are processed in batches in the background, that is, when no routing protocol traffic is being processed. The **flash** option is used to control the number and type of routes processed during a flash update.

Defaults

```
kernel { flash limit 20 type interface; };
```

Context

kernel statement

Examples

flash update up to 40 routes at a time. All types of routes are updated:

```
kernel { flash 40 type all ; } ;
```

See Also

background on page 327

“Chapter 18 Kernel Interface” on page 95 of *Configuring GateD*

kernel on page 327

kernel

Name

kernel - controls how GateD interfaces with the kernel

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 ) ]
    ; ]
} ;
```

Parameters

options
remnantholdtime
routes *number*
flash
background
traceoptions

Description

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 now done in batches. The size of these batches can be controlled by the tuning parameters described in the **routes**, **flash**, and **background** sections, 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.

Options **nochange** and **noflushatexit** are disabled by default.

Defaults

```
kernel {
    flash limit 20 type interface;
    routes -1;          # no limit on kernel-installed routes
    background limit 120 priority lower;
    remnantholdtime 180;
    traceoptions none;
};
```

Context

global

Examples

```
kernel {
    options nochange;
    flash limit 40 type all;
    routes 5000;
    background limit 500 priority flash;
    remnantholdtime 7:0;
};
```

See Also

background on page 327

“Chapter 18 Kernel Interface” on page 95 of *Configuring GateD*

flash on page 329

`options` on page 334

`remnantholdtime` on page 336

`routes` on page 337

`traceoptions` on page 339

options

Name

options - specifies the kernel statement options

Syntax

```
options ( nochange | noflushatexit ) ;
```

Parameters

nochange - On systems supporting the routing socket, **nochange** ensures that change operations will not be performed (only deletes and adds will). **nochange** is useful on early versions of the routing socket code where the change operation was broken.

noflushatexit - During normal shutdown processing, GateD deletes all routes that do not have a retain indication from the kernel forwarding table. **noflushatexit** prevents route deletions at shutdown.

During a GateD shutdown/restart sequence, it may be desirable to keep the routes that existed at the time of the GateD shutdown in the kernel forwarding table. This enables the router to continue forwarding packets while GateD is being restarted.

After a restart, the protocol modules are given a short amount of time (currently 3 minutes) to determine their routes. After 3 minutes, all residual routes not re-established by the protocol modules are flushed.

There are four conditions under which GateD does not flush a route:

- interface routes
- static routes with **retain** keyword
- **noflushatexit**
- routes with static bit set

noflushatexit is handy on systems with thousands of routes. Upon startup, GateD will notice which routes are in the kernel forwarding table and not add them back.

Description

options specifies the kernel statement options.

Defaults

nochange and **noflushatexit** are disabled.

Context

kernel statement

Examples

Prevent FIB change calls.

```
kernel { options nochange; } ;
```

See Also

“Chapter 18 Kernel Interface” on page 95 of *Configuring GateD*

kernel on page 327

remnantholdtime on page 336

remnantholdtime

Name

remnantholdtime - sets the default remnant hold time

Syntax

remnantholdtime *time*

Parameters

time - The remnant hold time must be between 0 and 900 seconds. It can be expressed as *seconds* or *minutes:seconds*.

seconds - time in seconds

minutes:seconds - time in minutes and seconds, separated by a colon

Description

When GateD starts up, it reads the kernel forwarding table and installs corresponding routes into GateD's routing table. These routes, with the exclusion of interface routes and routes configured via the UNIX route command, are called remnants. Remnant routes are timed out after the specified interval, or as soon as a more attractive route is learned. This method allows forwarding to occur while the routing protocols start learning routes. Setting **remnantholdtime** to 0 disables this timer, causing remnant routes to be kept indefinitely.

Defaults

Delete remnant routes after 180 seconds:

```
kernel {remnantholdtime 180;};
```

Context

kernel statement

Examples

Set the remnant hold time to 5 minutes.

```
kernel { remnantholdtime 5:0 ; } ;
```

See Also

"Chapter 18 Kernel Interface" on page 95 of *Configuring GateD*

kernel on page 327

routes

Name

routes - limits the number of routes in the kernel's forwarding table

Syntax

routes *number*

Parameters

number - an integer specifying the maximum number of routes that GateD will place in the kernel's routing table. *number* is essentially unlimited. It must be in the range 0 to 4,294,967,295. Note that *number* is unsigned. Specifying -1 results in the maximum 32-bit unsigned number, which is mentioned above.

Description

On some systems, kernel memory is at a premium. With **routes**, a limit can be placed on the maximum number of routes GateD will install in the kernel. This discussion is concerned with three types of routes:

- interface routes: routes defined by an **interface** statement (includes UNIX 'ifconfig' and 'route' generated routes)
- interior routes: routes within the domain
- exterior routes: routes exterior to the domain

Normally, GateD adds, changes, or deletes routes in interface/interior/exterior order. That is, GateD queues interface routes first, followed by interior routes, followed by exterior routes, and then processes the queue from the beginning. When the route limit is reached, GateD must ensure that interface/interior/exterior route preferences are followed. This is accomplished by first deleting kernel-based routes and then turning queued changes into adds. Finally, the list of active routes in the RIB is processed in interface/internal/external order, until the route limit is reached.

Defaults

There is no limit on kernel-installed routes.

```
kernel { routes -1; } ;
```

Context

kernel statement

Examples

Limit the number of kernel routes to 1000.

```
kernel { routes 1000; } ;
```

See Also

"Chapter 18 Kernel Interface" on page 95 of *Configuring GateD*

kernel1 on page 327

traceoptions

Name

traceoptions - enable tracing for kernel-related activities

Syntax

```
traceoptions none;
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 ) ];
```

Parameters

none - Do not trace any kernel packets.

replace - specifies to start tracing by truncating an existing file. The default is to append to an existing file.

size *tracesize* [**k** | **m**] **files** *tracefiles* - Limits the maximum size of the trace file to the specified *tracesize* (minimum 10k). When the trace file reaches the specified size, the file is renamed to file.0, then file.1, then file.2, up to the maximum number of files. (The minimum specification is 2.)

k - specifies the file size in kilobytes

m - specifies the file size in megabytes

files *tracefiles* - specifies the maximum number of files

nostamp - specifies that a timestamp should not be prepended to all trace lines

except - The listed options that follow are not traced.

trace_global_options - global trace options as defined in "Chapter 4 Trace Statements" on page 15 of Configuring GateD

trace_protocol_options - one or more of the following options:

Although the kernel interface isn't technically a routing protocol, in many cases it is handled as one. The following two *trace_options* can be entered from the command line because the code that uses them is executed before the trace file is parsed.

symbols - Trace symbols, which are read from the kernel, by **nlist()** or similar interface. 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 - Trace **iflist**, the interface list **scan**. **iflist** is useful when entered from the command line, because the first interface list scan is performed before the configuration file is parsed.

The following *trace_protocol_options* may only be specified in the configuration file. They are not valid from the command line.

remnants - trace remnants, which specify routes read from the kernel when GateD starts

request - trace requests that specify to add, delete, or change routes in the kernel forwarding table

The following general option and packet *trace_protocol_options* apply only on systems that use the routing socket to exchange routing information with the kernel. They do not apply on systems that use the old *BSD 4.3 ioctl()* interface to the kernel.

info - Trace info messages, which are messages received from the routing socket, such as TCP lossage, routing lookup failure, and route resolution requests. GateD does not currently process these messages, but logs the information if requested.

trace_protocol_packets - [**detail**] [**send** | **receive**] (**packets** | **routes** | **redirect** | **interface** | **other**) as defined below:

detail - Use detailed packet tracing.

send - Trace only kernel packets sent.

receive - Trace only kernel packets received.

packets - Trace all kernel packet types.

redirect - Trace only kernel redirect packets.

routes - Trace routes that are exchanged with the kernel, including add, delete, or change messages and add, delete, or change messages received from other processes.

redirect - Trace redirect messages, which are received from the kernel.

interface - Trace interface status messages that are received from the kernel. These are supported only on systems with networking code derived from *BSD 4.4*

other - Trace other messages that are received from the kernel, including those mentioned in the info type above. This option is currently not being used and is reserved for future use.

Description

Trace statements control tracing options. The trace options specified here are used to enable tracing of kernel-related event processing. The kernel events that can be traced include routing socket messages and kernel forwarding table updates.

Defaults

The parameters described above are disabled by default.

Context

kernel statement

Examples

Traces routes exchanged with the kernel.

```
kernel { traceoptions routes ; } ;
```

See Also

“Chapter 18 Kernel Interface” on page 95 of *Configuring GateD*
kernel on page 327

Chapter 15

Static Routes

as

Name

as - associates the specified autonomous system number with the static route

Syntax

```
as autonomous_system ;
```

Parameters

autonomous_system - the autonomous system number from which routes are to be learned

Description

as associates the specified autonomous system number with the static route.

Defaults

none

Context

static statement

Examples

```
static {  
    host 211.14.165.243  
    gateway 12.17.99.45  
    as 201;  
};
```

See Also

static on page 357

gateway on page 346

blackhole

Name

blackhole - causes this route to be installed as a blackhole route

Syntax

blackhole ;

Parameters

none

Description

blackhole is a mechanism enabling the router to refuse to route various prefixes. The prefixes are represented as routes that are not reachable, called unreachable routes. A blackhole route is the same as a reject route except that unreachable messages are not generated. Specifying **blackhole** causes this route to be installed as a blackhole route. **blackhole** should only be used with systems based on *BSD 4.3 Tahoe* or earlier that have installed a reject or blackhole pseudointerface. For interface routes, **blackhole** specifies that the address of the interface that matches these criteria is to be used as the local address when installing reject routes in the kernel.

Defaults

disabled

Context

static statement

Examples

```
static {  
    192.0.2.0/24  
    interface en1  
    blackhole;  
};
```

See Also

static on page 357

default

See **static** on page 357

gateway

Name

gateway - defines static routes through a gateway

Syntax

```
gateway gateway_list
```

Parameters

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

Description

static gateway statements define static routes through a gateway. The **static gateway** statement defines a route to a destination host, to a subnet, or to the default prefix (0.0.0.0/0). Static gateway routes are installed in the kernel forwarding information base when one or more of the listed gateways are available. The gateway can be directly attached to an interface, or indirectly reachable via directly connected routers. Some versions of the UNIX operating system support equal-cost multipath next hops, supporting load sharing among the next hops. The number of multipath destinations supported by the UNIX kernel is a compile-time constant. If more than one specified gateway is available, and the kernel supports multipath destinations, multiple routes to a destination will be installed.

Defaults

none

Context

static statement

Examples

Example 1

```
static {  
    host 211.14.165.243           # host reachable via  
    gateway 12.17.99.45;         # specified gateway  
};
```

Example 2

```
static {  
    default  
    gateway 196.44.21.12;        # default route through gateway;  
};
```

Example 3

```
static {  
    242.2.218.5 mask 255.255.254.0    # define 23-bit subnet address  
    gateway 12.17.99.45;              # reachable via gateway  
};
```

See Also

`static` on page 357

`default` on page 345

host

See **static** on page 357

interface

Name

interface - specifies an interface or list of interfaces to associate with a static route

Syntax

```
interface interface
where interface is
    ( name | address | local address | remote address )
interface interface_list
where interface_list is
    all | ( interface ... )      # a list of interfaces
```

Parameters

all - all available interfaces

name - the name of an interface or a host DNS name to use as the unique address of the interface

address - the unique address of the interface

local *address* - the local address of the interface

remote *address* - the remote address of the interface

Description

For **static gateway**, **interface** is followed by an *interface_list*, which specifies the list of acceptable interfaces to be used to reach the gateways specified in *gateway_list*. The gateways are examined in the order they were specified to see if they are reachable by an interface included in *interface_list*. The first up to **RT_N_MULTIPATH** (compile-time constant) gateways reachable via an interface allowed by *interface_list* become the nexthops for the static route. *interface_list* can contain wildcards (i.e., a physical interface name without trailing digits).

For the static interface case, only a single interface can be specified as the *interface* associated with the static route. The route will be eligible to become active only if the associated interface is up. The nexthop address will be the local interface address.

Defaults

There is no default for the **static interface** case. In the **static gateway** case, the interface for a given gateway defaults to the interface via which the specified gateway address is reachable.

Context

static statement

Examples

Example 1

```
static {  
    192.0.2.0/24  
    interface en1  
    blackhole;  
};
```

Example 2

```
static {  
    default  
    gateway 196.44.21.12  
    interface ex1;  
}
```

See Also

[gateway](#) on page 346

multicast

Name

`multicast` - loads this route in the multicast RIB

Syntax

`multicast ;`

Parameters

none

Description

This route will be loaded in the multicast RIB. Static routes are installed into the multicast RIB only by specification.

Defaults

unicast RIB

Context

`static` statement

Examples

```
static {  
    host 2.6.5.35  
    gateway 8.9.7.93  
    multicast;  
} ;
```

See Also

`unicast` on page 359

`static` on page 357

`gateway` on page 346

noinstall

Name

noinstall - specifies that this static route is not to be installed in the kernel forwarding table

Syntax

```
noinstall ;
```

Parameters

none

Description

Normally, the route with the lowest preference is installed in the kernel forwarding table and is the route exported to other protocols. When **noinstall** is specified on a static route, it will not be installed in the kernel forwarding table when it is active, but it will still be eligible to be exported to other protocols.

Defaults

Active static routes are installed in the kernel forwarding table.

Context

static statement

Examples

```
static {  
    route 2.6.5.35  
    gateway 8.9.7.93  
    noinstall;  
} ;
```

See Also

static on page 357

gateway on page 346

preference

Name

preference - used to select the best route, when multiple routes exist for the same destination

Syntax

preference *preference*

Parameters

preference - Preferences are in the range 0 to 255, inclusive, with 0 being the lowest (best) preference a route can have.

Description

Multiple routes can exist for the same destination. When multiple routes exist for the same destination, the route's preference is used to select the best route. **preference** overrides the default preference for this static route.

Defaults

preference 60;

Context

static statement

Examples

Example 1

```
static {  
    host 211.14.165.243  
    gateway 12.17.99.45  
    preference 30;  
};
```

Example 2

```
static {  
    199.14.128.0  
    mask 255.255.224.0  
    interface en1  
    preference 15;  
} ;
```

See Also

`static` on page 357

“Interface Statement” on page 23 of *Configuring GateD*

reject

Name

reject - enables the router to refuse to route to various prefixes

Syntax

```
reject ;
```

Parameters

none

Description

This is a mechanism enabling the router to refuse to route to various prefixes. The prefixes are represented as routes that are not reachable, called unreachable routes. Instead of forwarding a packet as a normal route, **reject** routes cause packets to be dropped and unreachable messages to be sent to the packet originators. Specifying **reject** causes this static route to be installed as a reject route. Not all kernel forwarding engines support reject routes.

For interface routes, **reject** specifies that the address of the interface that matches **interface** will be used as the local address when installing reject routes in the kernel.

Defaults

disabled

Context

static statement

Examples

```
static {  
    host 192.0.2.0  
    gateway 172.31.255.255  
    reject;  
} ;
```

See Also

static on page 357

interface on page 349

gateway on page 346

"Interface Statement" on page 23 of *Configuring GateD*

blackhole on page 344

retain

Name

retain - prevents specific static routes from being removed

Syntax

retain ;

Parameters

none

Description

Normally, GateD removes all routes except interface routes and kernel routes with the RTF_STATIC bit set from the kernel forwarding table during a graceful shutdown. The retain option is used to prevent specific static routes from being removed. Retain ensures that some routing is available when GateD is not running.

Defaults

Static routes are not retained at graceful shutdown.

Context

static statement

Examples

```
static {  
    route 2.6.5.35  
    gateway 8.9.7.93  
    retain;  
} ;
```

See Also

static on page 357

gateway on page 346

interface on page 349

static

Name

static - defines static routes through a gateway

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 ] ;
};
```

Parameters

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
```

Description

static gateway route statements define static routes through a gateway. 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.

Defaults

Static routes do not exist by default.

Context

global

Examples

```
static {  
    host 211.14.165.243      # host reachable via  
    gateway 12.17.99.45;    # specified gateway  
};  
  
static { 199.14.128.0 mask 255.255.224.0 interface en1 ; } ;
```

See Also

`interface` on page 349

`gateway` on page 346

unicast

Name

`unicast` - specifies that the route will be loaded into the unicast RIB

Syntax

`unicast ;`

Parameters

none

Description

This route will be loaded in the unicast RIB. This is useful when a route is being installed into both the unicast and multicast RIBs.

Defaults

By default, all static routes are loaded into the unicast RIB.

Context

`static` statement

Examples

```
static {  
    host 2.6.5.35  
    gateway 8.9.7.93  
    unicast;  
} ;
```

See Also

`multicast` on page 351

`interface` on page 349

`gateway` on page 346

Chapter 16

Distance Vector Multicast Routing Protocol (DVMRP)

defaultmetric

Name

defaultmetric - specifies the default value for the interface metric

Syntax

```
defaultmetric metric ;
```

Parameters

metric - numeric value, ranging from 1 to 32

Description

defaultmetric specifies the metric applied to any interface that does not explicitly have a metric set with the **dvmrp interface, metric** parameter.

Default

```
defaultmetric 1 ;
```

Context

dvmrp statement

Examples

This example configures two interfaces eth0 and eth1. eth0 has an explicit metric given in its **interface** statement. eth1 does not, so the default metric value of 10 will apply to eth1, but not to eth0.

```
dvmrp yes {  
    interface eth0 {  
        metric 5;  
    };  
    interface eth1;  
    defaultmetric 10;  
};
```

See Also

“Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)” on page 103 of *Configuring GateD*

metric on page 368

disable

Name

`disable` - disables DVMRP on this interface

Syntax

```
disable ;
```

Parameters

none

Description

`disable` explicitly disables the interfaces provided in the interface list.

Default

```
enable ;
```

Context

`dvmrp interface` statement

Examples

Enable all interfaces of type eth, but specifically disable eth0.

```
dvmrp on {  
    interface eth;  
    interface eth0 {  
        disable;  
    };  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

`enable` on page 366

`routing-only` on page 373

dvmrp

Name

dvmrp - enables or disables the DVMRP protocol

Syntax

```
dvmrp off ;  
dvmrp routing-only {dvmrp parameters} ;  
dvmrp on {dvmrp parameters} ;
```

Parameters

dvmrp parameters

Description

If enabled, DVMRP will default to enabling all interfaces that are multicast capable. **dvmrp routing-only** specifies that DVMRP will be used only to propagate the multicast RIB, but that it will not be used for tree construction. **dvmrp routing-only** may be used to let DVMRP carry the multicast RIB, which is then used by PIM-SM. *dvmrp parameters* include all the parameters in this section.

Default

```
dvmrp off ;
```

Context

global

Examples

Example 1

This explicitly turns DVMRP off. However, because DVMRP is off by default, it is unnecessary.

```
dvmrp off;
```

Example 2

This turns DVMRP on and configures all multicast-capable interfaces.

```
dvmrp on;
```

Example 3

This turns DVMRP on and configures only two interfaces.

```
dvmrp on {  
    interface eth0 eth1;  
};
```

Example 4

This turns on DVMRP routing only on all multicast-capable interfaces.

```
dvmrp routing-only;
```

Example 5

This turns on DVMRP routing on only two interfaces.

```
dvmrp routing-only {  
    interface eth0 eth1;  
};
```

See Also

bgp on page 227

“Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)” on page 103 of *Configuring GateD*

export on page 623

import on page 605

isis on page 153

ospf on page 95

enable

Name

enable - enables DVMRP on this interface

Syntax

enable ;

Parameters

none

Description

enable explicitly enables the interfaces provided in the interface list.

Default

enable ;

Context

dvmrp interface statement

Examples

Disable all interfaces of type eth, but specifically enable eth0.

```
dvmrp on {  
    interface eth {  
        disable;  
    };  
    interface eth0 {  
        enable;  
    };  
};
```

See Also

“Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)” on page 103 of *Configuring GateD*

disable on page 363

routing-only on page 373

interface

Name

interface - explicitly configures one or more interfaces for DVMRP

Syntax

```
interface interface_list { dvmrp interface parameters };
```

Parameters

interface_list - a list of one or more interface names, interface wildcards, or IP addresses

Description

interface configures interfaces for use by DVMRP. If no interfaces are explicitly configured, then all multicast-capable interfaces are implicitly enabled; however, if even one interface is explicitly configured, then no interfaces are implicitly enabled.

Default

```
interface all;
```

Context

dvmrp statement

Examples

All multicast-capable interfaces are implicitly enabled.

```
dvmrp on;
```

Explicitly configure two interfaces, enabling the first and disabling the second.

```
dvmrp on {  
    interface eth0 {  
        enable;  
    };  
    interface 192.168.0.1 {  
        disable;  
    };  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

dvmrp on page 364

metric

Name

metric - explicitly sets a metric for an interface

Syntax

```
metric metric ;
```

Parameters

metric - an integer value ranging from 1 to 32, inclusive

Description

metric sets the metric to be used on the interface. This value is added to all routes learned from neighbors through this interface.

Default

```
metric 1;
```

Context

dvmrp interface statement

Examples

```
dvmrp on {  
    interface eth0 {  
        metric 3;  
    };  
    interface eth1 {  
        metric 5;  
    };  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

defaultmetric on page 361

interface on page 367

nodvmrpout

Name

`nodvmrpout` - tells DVMRP to just listen on an interface

Syntax

```
nodvmrpout ;
```

Parameters

none

Description

`nodvmrpout` disables DVMRP as a speaker on an interface, although it will listen and accept routes on the interface.

Default

off

Context

`dvmrp interface` statement

Examples

DVMRP will normally listen and report on all interfaces except eth1.

```
dvmrp on {  
    interface eth {  
        enable ;  
    } ;  
    interface eth2 {  
        enable ;  
        nodvmrpout ;  
    } ;  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

`dvmrp` on page 364

noretransmit

Name

`noretransmit` - specifies to refrain from resending DVMRP prune packets

Syntax

```
noretransmit ;
```

Parameters

none

Description

`noretransmit` configures GateD to not perform the exponential backoff prune retransmission. After the transmission of the first prune, no additional prunes will be transmitted on reception of data until the prune lifetime has expired.

Default

off

Context

`dvmrp interface` statement

Examples

```
dvmrp on {  
    interface eth0;  
    interface eth1 {  
        noretransmit;  
    };  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

`dvmrp` on page 364

preference

Name

preference - sets the value that GateD uses for DVMRP routes in the active route selection process

Syntax

```
preference pref ;
```

Parameters

pref - an integer value ranging from 0 to 255, inclusive

Description

preference sets the value that GateD uses for DVMRP routes in the active route selection process. A route from a protocol with a lower preference value is selected over a route from a protocol with a higher preference value.

Default

```
preference 70 ;
```

Context

dvmrp statement

Examples

```
dvmrp on {  
    preference 10;  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

dvmrp on page 364

prune-lifetime

Name

prune-lifetime - specifies the maximum default lifetime of prunes in seconds

Syntax

```
prune-lifetime time;
```

Parameters

time - number of seconds as an integer value ranging from 1 to 2,147,483,648, inclusive

Description

prune-lifetime configures the maximum value to be placed into a prune message. The actual lifetime value is the minimum of all the downstream prunes for the source and a randomized value that falls between one-half the prune lifetime and the prune lifetime. The value is in seconds.

Default

```
prune-lifetime 7200;
```

Context

dvmrp statement

Examples

Set the maximum prune lifetime to 1 hour.

```
dvmrp on {  
    prune-lifetime 3600;  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

dvmrp on page 364

routing-only

Name

`routing-only` - specifies to do DVMRP route exchange on this interface only

Syntax

```
routing-only ;
```

Parameters

none

Description

`routing-only` configures the interfaces provided in the interface list to perform only DVMRP route exchange. The DVMRP multicast delivery tree-building operations will not be performed on this interface.

Default

```
enable ;
```

Context

`dvmrp` statement

`dvmrp interface` statement

Examples

```
dvmrp on {  
    interface eth0;  
    interface eth1 {  
        routing-only;  
    };  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

`disable` on page 363

`enable` on page 366

traceoptions

Name

traceoptions - specifies the tracing options for DVMRP

Syntax

```
traceoptions trace_options ;
```

Parameters

packets - Trace all DVMRP packets.

probe - Trace DVMRP probe packets.

report - Trace DVMRP route report packets.

mapper - Trace DVMRP neighbor and neighbor2 packets.

prune - Trace DVMRP prune packets.

graft - Trace DVMRP graft and graft ack packets.

detail - must be specified before **send** or **recv**. 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.

send or **receive** - Limit the tracing to packets sent or received, respectively. If neither is specified, both sent and received packets will be traced.

Description

Packet tracing options can be modified with **detail**, **send**, or **recv**. **packets** traces all DVMRP packets. **probe** traces all DVMRP router probe packets. **report** traces all DVMRP route report packets. **mapper** traces all DVMRP neighbor and neighbor 2 packets. **prune** traces all DVMRP prune packets. **graft** traces all DVMRP graft and graft ack packets.

Default

DVMRP not traced by default

Context

dvmrp statement

Examples

Example 1

Trace all packets sent.

```
dvmrp on {  
    traceoptions packets send;  
};
```

Example 2

Trace all probes received.

```
dvmrp on {  
    traceoptions probe recv;  
};
```

Example 3

Trace all prunes and graft/graft ack packets.

```
dvmrp on {  
    traceoptions prune graft;  
};
```

See Also

"Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)" on page 103 of *Configuring GateD*

`dvmrp` on page 364

tunnel-compatible

Name

`tunnel-compatible` - configures GateD to perform old-style DVMRP tunnel encapsulation

Syntax

`tunnel-compatible ;`

Parameters

none

Description

`tunnel-compatible` configures GateD to perform old-style DVMRP tunnel encapsulation. In old-style tunnel encapsulation, DVMRP control messages are not ip-ip encapsulated, but merely unicasted to the tunnel endpoint.

Default

off

Context

`dvmrp interface` statement

Examples

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 on {
    interface 10.1.16.4;
};
static {
    10.1.16.0 masklen 24 gw 10.1.25.14;
};
```

Example 2

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

```
};  
dvmrp on {  
    interface 10.1.16.4 {  
        tunnel-compatible;  
    };  
};  
static {  
    10.1.16.0 masklen 24 gw 10.1.25.14;  
};
```

See Also

“Chapter 20 Distance Vector Multicast Routing Protocol (DVMRP)” on page 103 of *Configuring GateD*

`dvmrp` on page 364

Chapter 17

Protocol Independent Multicast (PIM)

assert-holdtime

Name

assert-holdtime - specifies the number of seconds that Assert state should be maintained in the absence of a refreshing Assert message

Syntax

```
assert-holdtime sec ;
```

Parameters

sec - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

When a PIM router receives an Assert message, it modifies the outgoing interface list for a (*,G) or (S,G) entry, as specified by the message. The lifetime of this modification is specified by the **assert-holdtime** statement. If another Assert message does not refresh the Assert state before the lifetime expires, then the outgoing interface list reverts to its previous state. The **assert-holdtime** statement can appear outside of a **sparse** clause, or it can appear within an **interface** statement. In the former case, the holdtime applies to all configured interfaces. In the latter case, the holdtime applies only to the interface(s) to which the **interface** statement refers. An **assert-holdtime** statement associated with an interface overrides any **assert-holdtime** statement that may appear outside of the **sparse** statements.

Defaults

```
assert-holdtime 180 ;
```

Context

pim statement

pim-sm (sparse) **interface** statement

Examples

Example 1

The following `pim` statement configures a PIM-SM component, "sm0" with an Assert hold-time of 20 seconds. This holdtime applies to all the interfaces configured for the component.

```
pim yes {  
    assert-holdtime 20;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", with interface fxp1 having an Assert holdtime of 45 seconds. Interface fxp0 will be configured with the default Assert holdtime of 20 seconds, specified by the `assert-holdtime` statement appearing outside of the `sparse` statement.

```
pim yes {  
    assert-holdtime 20;  
    sparse "sm0" {  
        interface fxp0 ;  
        fxp1 {  
            assert-holdtime 45;  
        }  
    };  
};
```

See Also

`pim` statement on page 414

boundary

Name

boundary - specifies that the associated interface is at a PIM domain boundary

Syntax

```
boundary ;
```

Parameters

none

Description

boundary specifies that the indicated interface(s) is (are) at a PIM domain boundary. PIM-SM Bootstrap Router (BSR) and Candidate-RP-Advertisement messages will not be sent or accepted over the associated interface(s). PIM Join/Prune messages however, are still exchanged. **boundary** makes it possible for adjacent but administratively separate PIM domains to be connected via MSDP, and thus for multicast group members in one domain to learn of and receive traffic from sources in another.

Defaults

no boundary specified for an interface

Context

pim-sm (sparse) **interface** statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", with a boundary on interface fxp1 and no boundary on interface fxp0.

```
pim yes {  
    assert-holdtime 20;  
    sparse "sm0" {  
        interface fxp0;  
        interface fxp1 {  
            boundary;  
        };  
    };  
};
```

See Also

pim statement on page 414

bsr

Name

bsr - specifies that this router should act as a Candidate Bootstrap Router (CBSR)

Syntax

```
bsr ( ( address | on ) [ priority pri ] ) | ( off )
```

Parameters

address - *address* must be a valid IPv4 address, host name, or interface name associated with one of the interfaces configured within the **sparse** statement in which the **bsr** statement appears.

pri - If a BSR priority is specified using the **priority** keyword, then *pri* must be an integer in the range 0 to 255, inclusive.

Description

GateD provides two mutually exclusive methods for RP set distribution: statically configured RPs and BSR. The BSR method is compliant with the mechanism described in draft-ietf-pim-sm-v2-new-01. The **bsr** statement enables the BSR method of RP set distribution.

A PIM BSR is responsible for distributing RP and group address information to its PIM domain. Multiple routers in a PIM-SM domain may be configured as CBSRs, and the PIM-SM protocol provides an election mechanism for selecting a BSR from the candidate pool. If **bsr off** is specified, or if the **bsr** statement is omitted, then this router is not eligible to become a BSR.

PIM-SM BSR messages contain a BSR address. If **bsr on** or **bsr yes** is specified, then the BSR address is chosen from one of the interfaces configured for the PIM-SM component. If one wishes to specify the exact address to be used as the BSR address, then **bsr address** should be used, where *address* is the desired IPv4 address.

PIM-SM BSR messages also contain a BSR priority, which can be used to bias the BSR election process. CBSRs with higher priorities are preferred. If two CBSRs have the same priority, then the CBSR with the larger IP address is preferred. The optional **priority pri** statement contained within the **bsr** statement specifies this CBSR's priority. If the **priority pri** statement is omitted, then the priority defaults to 0.

Defaults

```
bsr no ;
```

Context

pim-sm (sparse) statement

Examples

Example 1

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CBSR with an interface chosen from one of the two configured interfaces. The advertised CBSR priority will be 0.

```
pim yes {
    sparse "sm0" {
        bsr on;
        interface 192.168.10.2 192.168.22.1;
    };
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CBSR with address 192.168.10.2. The advertised CBSR priority will be 0.

```
pim yes {
    sparse "sm0" {
        bsr 192.168.10.2;
        interface 192.168.10.2 192.168.22.1;
    };
};
```

Example 3

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CBSR with address 192.168.10.2. The advertised CBSR priority will be 10.

```
pim yes {
    sparse "sm0" {
        bsr 192.168.10.2 {
            priority 10;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};
```

See Also

`pim` statement on page 414

`crp` statement on page 388

static-rp statement on page 425

bsr-holdtime

Name

bsr-holdtime - specifies the time after which the elected Bootstrap Router (BSR) will be assumed unreachable when bootstrap messages are not received from it

Syntax

```
bsr-holdtime secs ;
```

Parameters

secs - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

bsr-holdtime specifies the time after which the elected BSR will be assumed unreachable when bootstrap messages have not been received from it. The recommended value for this parameter is $2 * \text{bsr-period} + 10$. The BSR mechanism implemented by GateD is described in draft-ietf-pim-sm-v2-new-01.

Note: There is a tying of **bsr-period** and **bsr-holdtime**. Configuring one away from default configures the other, unless it too is explicitly configured.

Defaults

```
bsr-holdtime 130 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. Once a BSR is elected, the BSR's holdtime will be 110 seconds. (The component itself is not configured to be a CBSR.)

```
pim yes {  
    sparse "sm0" {  
        bsr-holdtime 110;  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

bsr-period statement on page 386

pim statement on page 414

bsr statement on page 382

bsr-period

Name

bsr-period - specifies the interval between originating bootstrap messages and should be equal to 60 seconds

Syntax

```
bsr-period secs ;
```

Parameters

secs - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

If a router is acting as the elected BSR for a PIM-SM domain, **bsr-period** specifies the number of seconds the router should wait between successive bootstrap message transmissions.

Note: There is a tying of **bsr-period** and **bsr-holdtime**. Configuring one away from default configures the other, unless it too is explicitly configured.

Defaults

```
bsr-period 60 ;
```

Context

pim-sm (bsr) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CBSR with address 192.168.10.2. If this router is elected to be the BSR for the PIM-SM domain, it will generate a BSR message every 45 seconds.

```
pim yes {  
    sparse "sm0" {  
        bsr 192.168.10.2 {  
            priority 10;  
            bsr-period 45;  
        };  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

bsr-holdtime statement on page 385

pim statement on page 414

bsr statement on page 382

crp

Name

crp - specifies that this router should act as a Candidate Rendezvous Point (CRP)

Syntax

```
crp ( address | on | yes )
```

Parameters

address - the IPv4 address, host name, or interface name to advertise in C-RP-Adv messages as the CRP address

Description

The PIM protocol allows for multiple routers to volunteer as the Rendezvous Point (RP) for a given multicast group. Such volunteers are called CRPs. If **crp off** or **crp no** is specified, or if the **crp** clause is omitted entirely, then this PIM router is not a CRP.

PIM Candidate-RP-Advertisement messages contain the address of the CRP. If **crp on** or **crp yes** is specified, then the CRP address is chosen from the list of addresses configured for the PIM-SM component. If you want to specify the exact address to be used as the CRP address, you must use **crp address** where address is the desired IPv4 address, host name, or interface name.

If the **crp** statement is specified without naming group addresses within curly braces, (for example, **crp on**) then the router will be a CRP for the group address range, 224/4. If group addresses (or address ranges) are named within curly braces, then the router will be a CRP only for the named addresses/ranges.

A router is chosen as the RP for a multicast group from the set of CRPs via a well-known hash function. A CRP's suitability for a given multicast group may be biased with a priority. When choosing an RP for a group from the set of CRPs, the hash function is computed for each member of the set of CRPs with the lowest priority for the group. The CRP yielding the highest hash value is selected as the RP for the group.

(Note the difference between CRP and BSR priorities. For BSR priorities, higher values are better.)

The **crp** statement allows one to specify an optional default priority as well as a priority associated with individual group addresses. Both priorities must be integers between 0 and 255, inclusive.

Defaults

```
crp off ;
```

Context

pim-sm (sparse) statement

Examples

Example 1

In the following example, a single PIM-SM component, "sm0" is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. Since no group ranges are specified, the router will advertise itself as a CRP for groups in the range 224/4 (for example, all multicast addresses). Finally, it will advertise itself with a priority of 2.

```
pim yes {
    sparse "sm0" {
        crp on {
            priority 2;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};
```

Example 2

In the following example, a single PIM-SM component, "sm0" is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. The router will advertise itself as a CRP for groups 224.1.2.3 and 224.1.2.4. The advertisement for group 224.1.3.4 will contain a priority of 1, whereas the advertisement of 224.1.2.4 will contain a priority of 2.

```
pim yes {
    sparse "sm0" {
        crp on {
            priority 2;
            group {
                224.1.2.3 priority 1;
                224.1.2.4;
            } ;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};
```

Example 3

Group ranges can be specified within the curly braces of the **crp** statement via the **group-address mask mask** OR **group-address masklen length** statements. In addition, an ASCII network name can be specified with the **host** keyword. These features are illustrated below. The router will advertise itself as a CRP with priority 2 for all groups in the range 224.0.1.0 to 224.0.1.3, as well as group 224.0.1.1 (ntp.mcast.net).

```
pim yes {  
    sparse "sm0" {  
        crp on {  
            priority 2;  
            224.0.1.1 masklen 30;  
            host ntp.mcast.net;  
        };  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

`pim` statement on page 414

`bsr` statement on page 382

`static-rp` statement on page 425

crp-adv-period

Name

crp-adv-period - sets the interval at which a Candidate Rendezvous Point (CRP) will send CRP Advertisement (Adv) messages to the Bootstrap Router (BSR)

Syntax

```
crp-adv-period crp-adv-periodsecs ;
```

Parameters

crp-adv-periodsecs - an integer between 0 and 65535, inclusive

Description

When using the BSR mechanism to distribute RP set information throughout a PIM-SM domain, CRPs must periodically send C-RP-Adv messages to the BSR. The **crp-adv-period** statement specifies the time, in seconds, between successive C-RP-Adv messages.

Defaults

```
crp-adv-period 60 ;
```

Context

pim-sm (crp) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CRP with address 192.168.10.2 by sending a C-RP-Adv message every 20 seconds.

```
pim yes {
    sparse "sm0" {
        crp 192.168.10.2 {
            crp-adv-period 20;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};
```

See Also

pim statement on page 414

crp statement on page 388

crp-holdtime

Name

crp-holdtime - specifies the holdtime, in seconds, advertised in Candidate Rendezvous Point Advertisement (C-RP-Adv) messages

Syntax

```
crp-holdtime secs ;
```

Parameters

secs - an integer between 0 and 65535, inclusive

Description

For CRPs, **crp-holdtime** specifies the holdtime advertised in C-RP-Adv messages, and is used by the Bootstrap Router (BSR) to time out RPs. The recommended value for this parameter is $2.5 * [\text{crp-adv-period}]$.

Defaults

```
crp-holdtime 150 ;
```

Context

pim-sm (crp) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces 192.168.10.2 and 192.168.22.1. The component will announce itself as a CRP with address 192.168.10.2. The holdtime advertised in its C-RP-Adv message will be 30 seconds.

```
pim yes {  
    sparse "sm0" {  
        crp 192.168.10.2 {  
            crp-holdtime 30;  
        };  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

pim statement on page 414

crp statement on page 388

dr-switch-immediate

Name

dr-switch-immediate - causes a Designated Router (DR) to initiate a switch to the Shortest Path Tree (SPT) for (S,G) upon receipt of the first data packet from source S

Syntax

```
dr-switch-immediate ;
```

Parameters

none

Description

The PIM-SM protocol allows a Rendezvous Point (RP) or a DR to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data via the SP tree. Two methods are available within GateD for deciding when an SP tree switch should be initiated. One of these methods is to initiate a switch to the SP tree for an (S,G) pair upon receipt of the first data packet from S addressed to G. The **dr-switch-immediate** statement causes a DR to initiate an SP tree switch upon receipt of the first packet from S addressed to G.

If the **dr-switch-immediate** statement does not appear anywhere within the **sparse** statement, then a switch to the SP tree is initiated when the traffic rate exceeds a threshold.

Defaults

The default is to switch to the SP tree when the traffic rate exceeds a threshold.

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If the router is the DR for local receivers, then the first time this router receives data from a new source, S, addressed to group G, it will initiate an SPT switch for (S,G). If the data arrive before the router is the DR for local receivers, then the switch will be initiated only when the router becomes the DR for local receivers.

```
pim yes {
    sparse "sm0" {
        interfaces fxp0 fxp1 ;
        dr-switch-immediate ;
    }
}
```

See Also

`threshold` statement on page 426

`threshold-rp` statement on page 430

`threshold-dr` statement on page 428

`pim` statement on page 414

`rp-switch-immediate` statement on page 422

group

Name

group - specifies a set of multicast groups and optional priorities for which the router will volunteer to be a Candidate Rendezvous Point (CRP)

Syntax

```
group { group-information } ;
```

Parameters

group-information - the groups for which the router is volunteering to be a CRP

Description

When using the BSR method of RP set distribution, CRPs will periodically send C-RP-Adv messages to the BSR. These messages specify a set of groups for which the CRP is volunteering to be an RP. In addition, a priority is associated with each set of groups and is used to decide which CRP will actually serve as the RP for a given group. The **group** statement optionally appears inside the **crp** statement and allows you to configure the groups for which the router is volunteering to be an RP, as well as the priorities associated with these groups.

Defaults

If the **group** statement is omitted from the **crp**, then the router will send C-RP-Adv messages indicating that it will serve as an RP for all multicast groups. The priority associated with a set of groups is determined by the **priority** statement.

Context

pim-sm (**crp**) statement

Examples

Example 1

In the following example, a single PIM-SM component, "sm0" is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. Since no group ranges are specified, the router will advertise itself as a CRP for groups in the range 224/4 (for example, all multicast addresses). Finally, it will advertise itself with a priority of 2.

```
pim yes {
    sparse "sm0" {
        crp on {
            priority 2;
        };
        interface 192.168.10.2 192.168.22.1;
```

```
    };  
};
```

Example 2

In the following example, a single PIM-SM component, "sm0", is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. The router will advertise itself as a CRP for groups 224.1.2.3 and 224.1.2.4. The advertisement for group 224.1.3.4 will contain a priority of 1, whereas the advertisement of 224.1.2.4 will contain a priority of 2.

```
pim yes {  
    sparse "sm0" {  
        crp on {  
            priority 2;  
            group {  
                224.1.2.3 priority 1;  
                224.1.2.4;  
            } ;  
        };  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

Example 3

Group ranges can be specified within the curly braces of the **crp** statement via the *group-address mask mask* or *group-address masklen length* statements. In addition, an ASCII network name can be specified with the **host** keyword. These features are illustrated below. The router will advertise itself as a CRP with priority 2 for all groups in the range 224.0.1.0 to 224.0.1.3, as well as group 224.0.1.1 (ntp.mcast.net).

```
pim yes {  
    sparse "sm0" {  
        crp on {  
            priority 2;  
            224.0.1.0 masklen 30;  
            host ntp.mcast.net;  
        };  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

crp statement on page 388

hello-holdtime

Name

hello-holdtime - specifies how long neighbors should wait for Hello messages before expiring the sender's neighbor state

Syntax

```
hello-holdtime sec ;
```

Parameters

sec - an integer between 0 to 65535, inclusive

Description

PIM Hello messages contain a holdtime specifying how long neighbors must wait for Hello messages before expiring the sender's neighbor state. **hello-holdtime** specifies the holdtime, in seconds, to advertise in Hello messages.

The **hello-holdtime** statement can appear both outside of a **sparse** statement and within an **interface** statement. If it appears outside of a **sparse** statement, then the specified value will be used for Hello messages transmitted via all interfaces configured into PIM components. If it appears inside of an **interface** statement, then the specified value overrides any previous specifications for the associated interfaces.

Defaults

```
hello-holdtime 105 ;
```

Context

pim statement

pim (sparse) **interface** statement

Examples

Example 1

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Each Hello message will contain a holdtime of 45 seconds.

```
pim yes {  
    hello-holdtime 45;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Hello messages sent via interface fxp0 will contain a holdtime of 45 seconds, while those sent via interface fxp1 will contain a holdtime of 20 seconds.

```
pim yes {  
    hello-holdtime 45;  
    sparse "sm0" {  
        interface fxp0;      # Hellos will have holdtime of 45 secs  
        interface fxp1 {    # Hellos will have a holdtime of 20 secs  
            hello-holdtime 20;  
        };  
    };  
};
```

See Also

`hello-interval` statement on page 401

`pim` statement on page 414

hello-interval

Name

hello-interval - specifies the frequency with which Hello messages are sent

Syntax

```
hello-interval sec ;
```

Parameters

sec - an integer between 0 and 65535, inclusive

Description

PIM routers periodically multicast Hello messages on each network to which they are connected to alert other routers to the presence of the sender. The **hello-interval** parameter specifies the time, in seconds, between successive Hello messages.

The **hello-interval** statement can appear both outside of a **sparse** statement and within an **interface** statement. If it appears outside of a **sparse** statement, then the value specifies the time between Hello messages sent on all interfaces configured into PIM components. If it appears inside of an **interface** statement, then the specified value overrides any previous specifications for the associated interfaces.

Defaults

```
hello-interval 30 ;
```

Context

pim statement

pim-sm (sparse) **interface** statement

Examples

Example 1

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Hello messages will be sent via interfaces fxp0 and fxp1 every 35 seconds.

```
pim yes {  
    hello-interval 35;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Hello messages will be sent via interface fxp0 every 45 seconds, while those sent via interface fxp1 will be sent every 20 seconds.

```
pim yes {  
    hello-interval 45;  
    sparse "sm0" {  
        interface fxp0;    # Hellos will be sent every 45 secs  
        interface fxp1 {   # Hellos will be sent every 20 secs  
            hello-interval 20;  
        };  
    };  
};
```

See Also

`hello-holdtime` statement page 397

`pim` statement on page 414

hello-priority

Name

hello-priority - specifies the priority used to determine Designated Forwarder (DF) and include in PIM Hello messages

Syntax

```
hello-priority pri ;
```

Parameters

pri - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

PIM Hello messages may contain a priority field that is used to elect a DF on a shared network. All Hello messages originated by GateD contain such a priority. DFs are responsible for encapsulating multicast data from local sources into PIM-SM Register messages and for unicasting them to the Rendezvous Point. The router with the highest priority wins the DF election. In the case of a tie, the router with the highest IP address wins.

If at least one neighbor on the network does not use Hello priorities, then election of a DF is carried out using only IP addresses, where the highest address wins.

The **hello-priority** parameter can appear both outside of a **sparse** statement and within an **interface** statement. If it appears outside of a **sparse** statement, then the value specifies the priority of Hello messages sent on all interfaces configured into PIM components. If it appears inside of an interface statement, then the specified value overrides any previous specifications for the associated interfaces.

Defaults

```
hello-priority 1 ;
```

Context

pim statement

pim-sm (sparse) **interface** statement

Examples

Example 1

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Hello messages sent via interfaces fxp0 and fxp1 will contain a priority of 2.

```
pim yes {  
    hello-priority 2;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

```
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces `fxp0` and `fxp1`. Hello messages sent via interface `fxp0` will have a priority of 10, while those sent via interface `fxp1` will have a priority of 2.

```
pim yes {  
    hello-priority 10;  
    sparse "sm0" {  
        interface fxp0;          # Hello priority of 10  
        interface fxp1 {        # Hello priority of 2  
            hello-priority 2;  
        };  
    };  
};
```

See Also

`pim` statement on page 414

interface

Name

interface - specifies the interfaces associated with a PIM-SM component and over which PIM-SM will be spoken

Syntax

```
interface interface-list [ { pim-sm_interface_parameters } ] ;
```

Parameters

interface-list - the list of interfaces, IP addresses, host names, or interface names with which the component should be associated and over which PIM-SM will be spoken

pim-sm_interface_parameters - refers to the PIM-SM specific parameters, described in this document, that may be configured at the granularity of an interface

Description

This statement specifies the list of interfaces associated with a PIM-SM component and over which PIM-SM will be spoken. An interface does not need to be up (or even exist) to be configured. When the interface becomes available, PIM-SM will begin running on the interface. If the **interface** statement is omitted, then PIM will not be spoken at all.

Defaults

```
interface all {disable;;}
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0, fxp1 and fxp2. Interfaces fxp0 and fxp1 are configured with the **pim** statement-specified Hello priority of 10. Interface fxp2 is configured with an interface-specific Hello priority of 2.

```
pim yes {
    hello-priority 10;
    sparse "sm0" {
        interface fxp0 fxp1;    # Hello priority of 10
        interface fxp2 {      # Hello priority of 2
            hello-priority 2;
        };
    };
};
```

See Also

`pim` statement on page 414

jp-holdtime

Name

jp-holdtime - specifies the Join/Prune holdtime that is advertised in PIM Join/Prune messages

Syntax

```
jp-holdtime sec ;
```

Parameters

sec - an integer between 0 and 65535, inclusive

Description

This statement specifies the holdtime that is advertised in PIM Join/Prune messages. Receivers must wait at least this long after receiving a Join/Prune message before deleting the Join/Prune state associated with the advertiser. The recommended value is $3.5 * \text{jp-interval}$.

The **jp-holdtime** statement can appear both outside of a **sparse** statement and within an **interface** statement. If it appears outside of a **sparse** statement, then the value specifies the holdtime of Join/Prune messages sent on all interfaces configured into PIM components. If it appears inside of an **interface** statement, then the specified value overrides any previous specifications for the associated interfaces.

Note: There is a tying of **jp-holdtime** and **jp-interval**. If the **jp-interval** is configured and the **jp-holdtime** is not, then the **jp-holdtime** is automatically set to $3.5 * \text{jp-interval}$.

Defaults

```
jp-holdtime 210 ;
```

Context

pim statement

pim-sm (sparse) **interface** statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Join/Prune messages sent via interfaces fxp0 and fxp1 will contain a holdtime of 30.

```
pim yes {
    jp-holdtime 30;
    sparse "sm0" {
        interface fxp0 fxp1; # J/P Holdtime of 30
    };
};
```

See Also

`jp-interval` statement on page 407

`pim` statement on page 414

jp-interval

Name

jp-interval - specifies the frequency with which Join/Prune messages are sent

Syntax

```
jp-interval sec ;
```

Parameters

sec - an integer between 0 and 65535, inclusive

Description

The Join/Prune state on an upstream neighbor must be refreshed by periodic Join/Prune messages. This parameter specifies the number of seconds between successive Join/Prune messages sent to upstream neighbors to maintain the neighbor's Join/Prune state.

The **jp-interval** statement can appear both outside of a **sparse** statement and within an **interface** statement. If it appears outside of a **sparse** statement, then the value specifies the frequency with which periodic Join/Prune messages are sent on all interfaces configured into PIM components. If it appears inside of an **interface** statement, then the specified value overrides any previous specifications for the associated interfaces.

Note: There is a tying of **jp-holdtime** and **jp-interval**. If the **jp-interval** is configured and the **jp-holdtime** is not, then the **jp-holdtime** is automatically set to $3.5 * \text{jp-interval}$.

Defaults

```
jp-interval 60 ;
```

Context

pim statement

pim-sm (sparse) **interface** statement

Examples

Example 1

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. Periodic Join/Prune messages are sent to neighbors on interfaces fxp0 and fxp1 every 30 seconds.

```
pim yes {  
    jp-interval 30;  
    sparse "sm0" {  
        interface fxp0 fxp1; # J/P's sent every 30 secs  
    };  
};
```

```
};
```

Example 2

The following `pim` statement configures a PIM-SM component, "sm0", containing interfaces `fxp0` and `fxp1`. Periodic Join/Prune messages sent via interface `fxp0` will be sent every 30 seconds, while those sent via interface `fxp1` will be sent every 10 seconds.

```
pim yes {  
    jp-interval 30;  
    sparse "sm0" {  
        interface fxp0;          # J/P Holdtime of 10  
        interface fxp1 {  
            jp-interval 10;  
        };  
    };  
};
```

See Also

`jp-holdtime` statement on page 405

`pim` statement on page 414

mrt-period

Name

mrt-period - specifies the number of seconds to wait between examinations of a PIM component's multicast routing table (MRT)

Syntax

```
mrt-period sec ;
```

Parameters

sec - an integer between 1 and 3600, inclusive

Description

A PIM component's MRT is examined periodically in order to remove entries that have been marked for deletion. **mrt-period** specifies the number of seconds to wait between examinations. This can be a computationally expensive operation if the number of entries is large.

Defaults

```
mrt-period 15 ;
```

Context

pim (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". The MRT for this component will be examined every 30 seconds.

```
pim yes {  
    mrt-period 30;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

See Also

mrt-spt-mult statement on page 410

mrt-stale-mult statement on page 412

pim statement on page 414

mrt-spt-mult

Name

mrt-spt-mult - together with the **mrt-period**, specifies the interval at which the data rate threshold for all S,G entries will be checked for a possible switch to the SP tree

Syntax

```
mrt-spt-mult m ;
```

Parameters

m - an integer between 1 and 100, inclusive

Description

The PIM-SM protocol allows a Rendezvous Point (RP) or a Designated Router (DR) to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data from the SP tree. Two methods are available within GateD for deciding when an SPT switch should be initiated. One of these methods involves setting a threshold, in average bytes per second. If the data rate of traffic received from S addressed to G exceeds this threshold, then a switch is initiated. The **mrt-spt-mult** statement specifies the interval over which the average bytes per second calculation is made. The interval is expressed as a multiple of the **mrt-period**, defined by the **mrt-period** statement.

Defaults

```
mrt-spt-mult 14 ;
```

Context

pim (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". The MRT for this component will be examined every 30 seconds. If this router is an RP or DR for group G, then all (S,G) entries on the RP tree, where S can be any source, will be examined every 60 seconds to determine whether an SPT switch should be initiated for S.

```
pim yes {  
    mrt-period 30;  
    sparse "sm0" {  
        mrt-spt-mult 2;  
        interface fxp0 fxp1;  
    };  
};
```

See Also

mrt-period statement on page 409

rp-switch-immediate statement on page 422

dr-switch-immediate statement on page 393

pim statement on page 414

threshold statement on page 426

threshold-rp statement on page 430

threshold-dr statement on page 428

mrt-stale-mult

Name

mrt-stale-mult - together with the **mrt-period** statement, specifies the minimum number of seconds that a source may be silent before its corresponding (S,G) entry can be timed out

Syntax

```
mrt-stale-mult m ;
```

Parameters

m - an integer between 1 and 100, inclusive

Description

When a source stops sending to a group, the corresponding (S,G) entry is said to have become "stale" and is now a candidate for deletion from the PIM multicast forwarding table (MRT). The PIM-SM protocol defines a Keep-Alive Timer (KAT) for each (S,G) entry, which is reset by the arrival of data from source S addressed to group G. If the KAT ever expires, and if other conditions are met, then the entry can be deleted. The **mrt-stale-mult** statement, together with the **mrt-period** statement, specifies the minimum number of seconds that a source must be silent before the entry is considered stale. The value, *m*, specified in the **mrt-stale-mult** statement indicates that a source must be silent for *m* * **mrt-period** in order to be declared stale.

Defaults

```
mrt-stale-mult 14 ;
```

Context

pim (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". The MRT for this component will be examined every 30 seconds. The first sampling could take place immediately after seeing the last packet, and the second sampling at the next interval. In this example, it takes a minimum of 30 seconds and a maximum of 60 seconds to be considered stale.

```
pim yes {  
    mrt-period 30;  
    sparse "sm0" {  
        mrt-stale-mult 2;  
        interface fxp0 fxp1;  
    };  
};
```

See Also

`mrt-spt-mult` statement on page 410

`mrt-period` statement on page 409

`pim` statement on page 414

pim

Name

`pim` - enables or disables the PIM protocol

Syntax

```
pim ( on | off )
```

Parameters

`on` or `off`

Description

The `pim` statement enables or disables the PIM protocol. If the `pim` statement is not specified, PIM will not run. All interfaces that will run PIM must be multicast capable and specified within the `sparse` statement in order to determine the mode and PIM component with which the interface will be associated.

Defaults

```
pim off ;
```

Context

global

Examples

Example 1

The following `pim` statement configures a PIM-SM component, "sm0".

```
pim yes {  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

Example 2

Assume that GateD was running with the `pim` statement given in Example 1, and you want to disable the "sm0" component but maintain the configuration information in the configuration file. Simply change `yes` to `no`, as in the example below, and reconfigure GateD.

```
pim no {  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

See Also

“Protocol Independent Multicast (PIM)” on page 105 of the *Configuring GateD Guide*

priority

Name

priority - specifies a priority to be associated with the groups for which a router is volunteering to be a Candidate Rendezvous Point (CRP)

Syntax

```
priority pri ;
```

Parameters

pri - an integer between 0 and 255, inclusive

Description

When using the BSR method of RP set distribution, CRPs will periodically send C-RP-Adv messages to the BSR. These messages specify a set of groups for which the CRP is volunteering to be an RP. In addition, a **priority** is associated with each set of groups and is used to decide which CRP will actually serve as the RP for a given group. The **priority** statement optionally appears inside the **crp** statement and allows you to configure the groups for which the router is volunteering to be an RP, as well as the priorities associated with these groups. The **priority** statement can appear inside of the **group** statement, in which case the **priority** is associated with a specific group range. It can also appear outside of the **group** statement, in which case the indicated **priority** applies to all group ranges inside the **group** statement that have no explicitly associated priorities.

Defaults

If the **priority** statement is omitted from the **crp** statement, then the router's priority defaults to 0 for groups for which it volunteers to be an RP.

Context

pim-sm (crp) statement

pim-sm (group) statement

Examples

Example 1

In the following example, a single PIM-SM component, "sm0", is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. Since no group ranges are specified, the router will advertise itself as a CRP for groups in the range 224/4 (for example, all multicast addresses). Finally, it will advertise itself with a priority of 2.

```
pim yes {  
    sparse "sm0" {  
        crp on {  
            priority 2;  
        }  
    }  
}
```

```

    };
    interface 192.168.10.2 192.168.22.1;
};
};

```

Example 2

In the following example, a single PIM-SM component, "sm0" is specified. The router will act as a CRP with the CRP address being chosen from one of the two interfaces configured within the **sparse** statement. The router will advertise itself as a CRP for groups 224.1.2.3 and 224.1.2.4. The advertisement for group 224.1.3.4 will contain a priority of 1, whereas the advertisement of 224.1.2.4 will contain a priority of 0.

```

pim yes {
    sparse "sm0" {
        crp on {
            priority 2;
            group {
                224.1.2.3 priority 1;
                224.1.2.4;
            } ;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};

```

Example 3

Group ranges can be specified within the curly braces of the **crp** statement via the *group-address mask mask* or *group-address masklen length* statements. In addition, an ASCII network name can be specified with the **host** keyword. These features are illustrated below. The router will advertise itself as a CRP with priority 2 for all groups in the range 224.0.1.0 to 224.0.1.3, as well as group 224.0.1.1 (ntp.mcast.net).

```

pim yes {
    sparse "sm0" {
        crp on {
            priority 2;
            224.0.1.0 masklen 30;
            host ntp.mcast.net;
        };
        interface 192.168.10.2 192.168.22.1;
    };
};

```

See Also

`crp` statement on page 388

`group` statement on page 395

probe-period

Name

probe-period - specifies the number of seconds prior to the RegisterStop timer expiry to send a null Register message to the Rendezvous Point (RP)

Syntax

```
probe-period secs ;
```

Parameters

secs - an integer between 0 and **reg-sup-timeout**, inclusive

Description

When PIM null Register messages are used, **probe-period** specifies the number of seconds prior to the RegisterStop timer expiry to send a null Register message to the RP. If a PIM RegisterStop message is received from the RP before the RegisterStop timer expires, the RegisterStop timer is reset, and the sending of encapsulating Register messages is delayed.

Defaults

```
probe-period 5 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is a DR for a group G, and has received a RegisterStop message from the RP for source S, then it will send a probe (a null Register message) 10 seconds prior to the expiration of the Register Stop timer.

```
pim yes {  
    probe-period 10;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

See Also

reg-sup-timeout statement on page 420

pim statement on page 414

reg-sup-timeout

Name

reg-sup-timeout - specifies the number of seconds between receiving a PIM RegisterStop message and allowing Register messages encapsulating multicast data to again be sent

Syntax

```
reg-sup-timeout secs ;
```

Parameters

secs - an integer between 1 and 3600, inclusive

Description

When a router receives a RegisterStop message from a Rendezvous Point (RP) for an (S,G) pair, it must stop sending multicast data encapsulated in Register messages for some period of time. Such a router is said to be "register-suppressed" for the (S,G) pair. This statement specifies the number of seconds for which the router remains register-suppressed. A lower value means that the RP receives more frequent bursts of encapsulated multicast data, while a higher value means a longer join latency for new receivers. (Note that if null Registers are sent **probe-period** seconds before the timeout, then Register bursts are prevented, and **reg-sup-timeout** may then be lowered to decrease join latency.)

Defaults

```
reg-sup-timeout 60 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is a Designated Router for a group G, and has received a RegisterStop message from the RP for an (S,G) pair, then it will wait 45 seconds before once again encapsulating data from S to group G in Register messages.

```
pim yes {  
    sparse "sm0" {  
        reg-sup-timeout 45;  
        interface fxp0 fxp1;  
    };  
};
```

See Also

probe-period statement on page 418

pim statement on page 414

rp-switch-immediate

Name

rp-switch-immediate - causes a Rendezvous Point (RP) to initiate a switch to the Shortest Path (SP) tree for (S,G) upon receipt of the first Register message encapsulating data from source S

Syntax

```
rp-switch-immediate ;
```

Parameters

none

Description

The PIM-SM protocol allows an RP or a Designated Router (DR) to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data from the SP tree. Two methods are available within GateD for deciding when an SP tree switch should be initiated. One of these methods is to initiate a switch to the SP tree for an (S,G) pair upon receipt of the first Register message containing data from S addressed to group G.

If the **rp-switch-immediate** statement does not appear anywhere within the **sparse** statement, then an active RP will initiate a switch to the SP tree when the traffic rate exceeds a threshold.

Defaults

The default is to switch when the traffic rate exceeds a threshold.

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is an RP for G, then the first time that the router receives a Register message encapsulating data from S addressed to group G, it will initiate a switch to the SP tree rooted at S.

```
pim yes {  
    sparse "sm0" {  
        rp-switch-immediate;  
        crp 192.168.10.2;  
        interface 192.168.10.2 192.168.22.1;  
    };  
};
```

See Also

`threshold` statement on page 426

`threshold-rp` statement on page 430

`threshold-dr` statement on page 428

`pim` statement on page 414

`dr-switch-immediate` statement on page 393

sparse

Name

sparse - configures a PIM-SM component

Syntax

```
sparse component_name { pimsm_statements } ;
```

Parameters

component_name - a string to identify the PIM-SM component

pimsm_statements - refers to the PIM-SM-specific parameters described in this document that can be configured within the **sparse** statement

Description

The **sparse** statement is used to configure a PIM-SM component. At the time of this writing, GateD supports the configuration of only a single PIM-SM component.

Note: Each **sparse** statement must also contain at least one **interface** statement.

Defaults

The default is to configure no PIM-SM components at all.

Context

pim statement

Examples

The following pim statement configures a PIM-SM component, "sm0". The component will control the running of PIM-SM over interfaces fxp0 and fxp1.

```
pim yes {  
    sparse "sm0" {  
        interfaces fxp0 fxp1 ;  
    }  
}
```

See Also

interface statement on page 403

static-rp

Name

static-rp - lets you statically configure a Rendezvous Point (RP) set

Syntax

```
static-rp group-address masklen length rp-address ;
```

Parameters

group-address - a valid IPv4 group address

length - a mask length between 4 and 32, inclusive

rp-address - a valid IPv4 host address specifying the RP serving the group prefix indicated by the *group-address* and *length* parameters

Description

GateD provides two mutually exclusive methods for RP set distribution: statically configured RPs and Bootstrap Router (BSR). The **static-rp** statement lets you statically configure an RP set. Multiple **static-rp** statements can be used to add elements to the set.

Defaults

The default is to use the BSR method of RP set distribution.

Context

pim-sm statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". The RP 192.168.10.4 is configured to serve multicast group 224.40.2.1, while 192.168.10.10 is configured to serve all other multicast groups.

```
pim yes {
    sparse "sm0" {
        interfaces fxp0 fxp1 ;
        static-rp 224.40.2.1 masklen 32 192.168.10.4 ;
        static-rp 224.0.0.0 masklen 4 192.168.10.10 ;
    }
}
```

See Also

bsr statement on page 382

crp statement on page 388

threshold

Name

threshold - specifies the threshold, in bytes per second, which, when exceeded for an (S,G) pair, initiates a switch to the Shortest Path (SP) tree

Syntax

```
threshold bps ;
```

Parameters

bps - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

The PIM-SM protocol allows a Rendezvous Point (RP) or a Designated Router (DR) to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data from the SP tree. Two methods are available within GateD for deciding when an SP tree switch should be initiated. One of these methods involves setting a threshold, in average bytes per second. If data received from S addressed to G exceeds this threshold, then a switch is initiated. The threshold statement specifies this threshold in average bytes per second. The interval over which the average-bytes-per-second calculation is made is specified by the **mrt-period** and **mrt-spt-mult** statements.

Defaults

```
threshold 1000 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is an RP for G, or a DR for the pair (S,G), and if the data from S addressed to G exceeds an average of 10 packets per second, then an SP tree switch will be initiated for the pair. The period over which the average will be calculated will be the **mrt-period** times the **mrt-spt-mult**, 60 seconds.

```
pim yes {
    mrt-period 30;
    sparse "sm0" {
        threshold 10;
        mrt-spt-mult 2;
        interface fxp0 fxp1;
    };
};
```

See Also

`mrt-spt-mult` statement on page 410

`mrt-period` statement on page 409

`threshold-dr` statement on page 428

`threshold-rp` statement on page 430

`pim` statement on page 414

`rp-switch-immediate` statement on page 422

`dr-switch-immediate` statement on page 393

threshold-dr

Name

threshold-dr - specifies the threshold, in bytes per second, for a Designated Router (DR), which, when exceeded for an (S,G) pair, initiates a switch to the Shortest Path (SP) tree

Syntax

```
threshold-dr bps ;
```

Parameters

bps - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

The PIM-SM protocol allows a Rendezvous Point (RP) or a DR to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data from the SP tree. Two methods are available within GateD for deciding when an SP tree switch should be initiated. One of these methods involves setting a threshold, in average bytes per second. If data received from S addressed to G exceeds this threshold, then a switch is initiated. The **threshold-dr** statement specifies this threshold in average bytes per second observed by a DR. The **mrt-period** and **mrt-spt-mult** statements specify the interval over which the average-bytes-per-second calculation is made.

Defaults

```
threshold-dr 1000 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is an RP for G, then the data from S addressed to G must exceed an average of 10 bytes per second before an SPT switch is initiated. If this router is a DR for the pair (S,G), then the same data must exceed an average of 20 bytes per second before an SPT switch is initiated. The period over which the average will be calculated will be the **mrt-period** times the **mrt-spt-mult**, 60 seconds.

```
pim yes {  
    mrt-period 30;  
    sparse "sm0" {  
        threshold 10;  
        threshold-dr 20;  
        mrt-spt-mult 2;  
        interface fxp0 fxp1;  
    }  
}
```

```
};  
};
```

See Also

`threshold` statement on page 426

`mrt-spt-mult` statement on page 410

`mrt-period` statement on page 409

`threshold-rp` statement on page 430

`pim` statement on page 414

`rp-switch-immediate` statement on page 422

`dr-switch-immediate` statement on page 393

threshold-rp

Name

threshold-rp - specifies the threshold, in bytes per second, for a Rendezvous Point (RP), which, when exceeded for an (S,G) pair, initiates a switch to the Shortest Path (SP) tree

Syntax

```
threshold-rp bps ;
```

Parameters

bps - an integer between 1 and $2^{32} - 1$ (4,294,967,295), inclusive

Description

The PIM-SM protocol allows a RP or a Designated Router (DR) to switch from receiving data from a source S sent to a group G via the RP tree, to receiving data from the SP tree. Two methods are available within GateD for deciding when an SP tree switch should be initiated. One of these methods involves setting a threshold, in average bytes per second. If data received from S addressed to G exceeds this threshold, then a switch is initiated. The **threshold-rp** statement specifies this threshold in average bytes per second observed by an RP. The **mrt-period** and **mrt-spt-mult** statements specify the interval over which the average-bytes-per-second calculation is made.

Defaults

```
threshold-rp 1000 ;
```

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". If this router is an RP for G, then the data from S addressed to G must exceed an average of 100 bytes per second before an SPT switch is initiated. If this router is a DR for the pair (S,G), then the same data must exceed an average of 2000 bytes per second before an SPT switch is initiated. The period over which the average will be calculated will be the **mrt-period** times the **mrt-spt-mult**, 60 seconds.

```
pim yes {  
    mrt-period 30;  
    sparse "sm0" {  
        threshold 2000;  
        crp fxp0 {  
            threshold-rp 100;  
        };  
        mrt-spt-mult 2;  
    }  
};
```

```
        interface fxp0 fxp1;  
    };  
};
```

See Also

`threshold` statement on page 426
`mrt-spt-mult` statement on page 410
`mrt-period` statement on page 409
`threshold-dr` statement on page 428
`pim` statement on page 414
`rp-switch-immediate` statement on page 422
`dr-switch-immediate` statement on page 393

traceoptions

Name

traceoptions - specifies the tracing options for PIM components

Syntax

```
traceoptions trace_options ;
```

Parameters

Trace options include:

packets - Trace all types of PIM packets.

assert - Trace PIM Assert packets.

bootstrap - Trace PIM-SM Bootstrap packets.

hello - Trace PIM Hello packets.

jp - Trace PIM Join/Prune packets.

register - Trace PIM-SM Register and RegisterStop packets.

graft - Trace PIM-DM Graft and GraftAck packets.

crp - Trace PIM-SM Candidate-RP-Advertisement packets.

debug - extra trace information of use mainly to developers

Description

traceoptions specifies the tracing options for all PIM components. By default, **traceoptions** is off. If you do specify **traceoptions**, then all global tracing options are inherited.

Defaults

inherited from global **traceoptions**

Context

pim statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0". The component will trace Hello, Register and RegisterStop messages to the log file, /var/tmp/gated.log.

```
pim yes {  
    traceoptions "/var/tmp/gated.log" replace register hello;  
    sparse "sm0" {  
        interface fxp0 fxp1;  
    };  
};
```

See Also

`pim` statement on page 414

wholepkt-checksum

Name

wholepkt-checksum - specifies that checksums in Register messages should be calculated over the entire encapsulated data packet, rather than just over the Register message header

Syntax

```
wholepkt-checksum ;
```

Parameters

none

Description

Previous versions of the PIM-SM specification had the checksum of Register messages calculated over the entire message, including encapsulated data. Some versions of Cisco's IOS perform this form of checksumming. The latest version of the specification states that the checksum should be calculated only over the Register message header, not any encapsulated data. This is GateD's default checksum method. Use the **wholepkt-checksum** to specify that checksums in Register messages should be calculated according to the old method.

Defaults

The default is to calculate checksums over the Register message header only.

Context

pim-sm (sparse) statement

Examples

The following **pim** statement configures a PIM-SM component, "sm0", containing interfaces fxp0 and fxp1. When encapsulating data from local sources in Register messages and sending them to the Rendezvous Point, the message checksum will be calculated over the entire encapsulated packet, rather than just over the Register message header.

```
pim yes {  
    sparse "sm0" {  
        interface fxp0 fxp1 ;  
        wholepkt-checksum ;  
    }  
}
```

See Also

pim statement on page 414

Chapter 18

Multi-Protocol - Border Gateway Protocol (MPBGP)

allow

Name

allow - permits peer connections from any addresses in the specified range (including single host or "all") of network and mask pairs

Syntax

```
[inet6] allow {  
  all ;  
  | host ipaddress ;  
  | classful_network ;  
  | network mask mask ;  
  | network masklen masklennumber ;  
  | network / masklennumber ;  
}
```

Parameters

classful_network - the IPv4 address of the network to be peered, in dotted-quad format (xxx.xxx.xxx.xxx)

network - the IPv4 or IPv6 address of the network to be peered

mask - the address mask (modification) in dotted-quad format (xxx.xxx.xxx.xxx)

masklennumber - the number of contiguous one bits at the beginning of the mask

ip_address - the IPv4 or IPv6 address of the host network

Description

allow permits peer connections from any addresses in the specified range of networks. Multiple networks may be specified in the **allow** clause, but must be separated with semi-colons.

All parameters for these peers must be configured on the **group** clause. The internal peer structures are created when an incoming open request is received, and destroyed when the connection is broken.

For more details on specifying the network/mask pairs, see “Route Filtering” on page 129 in *Configuring GateD*.

Default

The default is an empty `allow` clause, allowing no non-explicitly configured peers.

Context

`mpbgp group type external statement`

`mpbgp group type internal statement`

`mpbgp group type routing statement`

Examples

Example 1

This allows anyone to peer with us and is not recommended.

```
allow {  
    all;  
} ;
```

Example 2

This allows a specific host to peer with us. Note that this is equivalent to specifying a `peer` statement without any options.

```
allow {  
    host 192.0.2.1 ;  
} ;
```

Example 3

This allows all hosts within a classful subnet.

```
allow {  
    10 ; # Allow all 10/8  
} ;
```

Example 4

This allows all hosts within a network:

```
allow {  
    192.0.2.0 mask 255.255.255.0 ;  
};
```

which is the same as:

```
allow {  
    192.0.2.0 masklen 24 ;
```

```
} ;
```

which is the same as:

```
allow {  
    192.0.2.0/24 ;  
}  
;
```

Example 5

You can specify more than one network in an **allow** clause.

```
allow {  
    192.0.2.0/24 ;  
    10.0.0.0/8 ;  
    host 172.16.0.1 ;  
}  
;
```

Example 6

IPv6 allow lists can also be specified.

```
inet6 allow {  
    all;  
};
```

Example 7

IPv6 and IPv4 lists can be used together to allow multiple address families.

```
inet6 allow {  
    fec0:0:0:70a::/64;  
};  
allow {  
    #IPv4 allow list  
    all;  
};
```

See Also

mpbgp on page 482

"Route Filtering" on page 129 in *Configuring GateD*

ascount

Name

ascount - configures the number of times that this router will prepend its autonomous system (AS) number to a route's AS Path when it sends the route to an external peer

Syntax

ascount *count*

Parameters

count - an integer between 1 and 25, inclusive

Description

ascount configures the number of times that this router will prepend its AS number to a route's AS Path when it sends the route to an external peer. Larger values are typically used to bias upstream peers' route selection. All things being equal, most routers will prefer routes with shorter AS Paths. Using **ascount**, the AS Path this router sends can be artificially lengthened.

The AS number that is prepended to the AS Path is the **localas** value, if specified, or the global **autonomoussystem** value.

ascount supersedes the deprecated **nov4asloop** option. Regardless of whether **nov4asloop** is set, this router will still send multiple copies of its own AS if the **ascount** option is set to something greater than 1.

Default

ascount 1

Context

mpbgp group type external statement
mpbgp peer statement

Examples

Example 1

The **ascount** is specified for a peer group.

[**autonomoussystem** and **routerid** omitted]

```
mpbgp on {  
    group type external peeras 64512 ascount 5 {  
        peer 192.0.2.2 ; # peer inherits ascount of 5 from group  
        peer 192.0.2.3 ; # peer inherits ascount of 5 from group  
    } ;  
} ;
```

[import and export statements omitted]

Example 2

The `ascount` is specified for a given peer.

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.2 ascount 5 ; # This peer has an ascount of 5  
        peer 192.0.2.3 ;           # This peer defaults to ascount 1  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

`localas` on page 469

`mpbgp` on page 482

“Route Filtering” on page 129 in *Configuring GateD*

caps

Name

caps - sets multiprotocol capabilities to be advertised to a group of peers

Syntax

```
caps { caps_list }  
caps no-caps
```

Parameters

caps_list - specify one or more of the following: **v4u**, **v4m**, **v4um**, **v6u**, **v6m**, **v6um** (separated by spaces)

Description

caps sets a list of multiprotocol capabilities to send to a group of peers (using the BGP Capability Advertisement mechanism defined in RFC 2842).

v4u, **v4m**, **v4um**, **v6u**, **v6m**, and **v6um** represent Address Family Identified/Subsequent Address Family Identifier (AFI/SAFI) pairs (see RFC 2858 - Multiprotocol Extensions for BGP-4). "**v4**" represents the IPv4 AFI. "**v6**" represents the IPv6 AFI. "**u**" represents unicast (SAFI 1), "**m**" represents multicast (SAFI 2), "**um**" represents unicast and multicast (SAFI 3). **no-caps** specifies that capabilities will not be understood. "**v6**" represents the IPv6 AFI.

All peers in this group must include all of the capabilities that are listed in the **caps** clause. If the remote peer is missing any of the capabilities specified in this clause, the peering session will not be established.

Default

none (capability announcement not performed)

Context

```
mpbgp group type external statement  
mpbgp group type internal statement
```

Example

```
[autonomoussystem and routerid omitted]  
mpbgp on {  
    group type internal peeras 65534 caps { v4u v4m v4um } {  
        # announce all ipv4 capabilities to this peer  
        peer 192.0.2.10;  
    } ;  
    group type external peeras 65533 caps { v4u } {  
        # announce only capability v4u to this peer  
        peer 192.0.2.11;  
    } ;  
}
```

```
    } ;  
    group type external peeras 65532 {  
        # no capabilities sent to this peer  
        peer 192.0.2.12;  
    } ;  
    group type external peeras 65531 caps { v6u } localv4addr 192.0.2.9 {  
        # announce the ipv6 unicast capability to this peer  
        peer fec0:0:0:ff::d;  
    } ;  
} ;
```

clusterid

Name

clusterid - specifies the route reflection cluster ID for MPBGP

Syntax

```
clusterid host-id ;
```

Parameters

host-id - a router ID, in dotted-quad format (xxxx.xxxx.xxxx.xxxx), used by route reflectors to prevent route propagation loops within the cluster

Description

clusterid specifies the route reflection cluster ID for MPBGP. (See "Route Reflection" on page 71 in *Configuring GateD*.) The cluster ID defaults to be the same as the router ID. (See "Router ID Syntax" on page 30 in *Configuring GateD* for more information about router IDs.) If a router is to be a route reflector, then a single cluster ID should be selected and configured on all route reflectors in the cluster. The only constraints on the choice of cluster ID are that:

- IDs of clusters within an autonomous system (AS) must be unique within that AS.
- The cluster ID must not be 0.0.0.0.

Choosing the cluster ID to be the router ID of one router in the cluster will always fulfill these criteria. If there is only one route reflector in the cluster, the **clusterid** setting may be omitted because the default will suffice.

Default

the globally configured **routerid**

Context

mpbgp statement

Examples

Example 1

The **clusterid** is specified.

[**autonomoussystem** omitted]

```
routerid 192.0.2.1 ;
```

```
mpbgp on {
```

```
    clusterid 192.0.2.254 ;
```

```
    group type internal peeras 65534 {
```

```
        peer 192.0.2.2 ;
```

```
        peer 192.0.2.3 ;
```

```

    } ;
# Routes received from clients will have the clusterid of
# 92.0.2.254 added to their BGP attributes.
    group type internal peeras 65534 reflector-client {
        peer 192.0.2.10 ;
        peer 192.0.2.11 ;
        peer 192.0.2.12 ;
        peer 192.0.2.13 ;
    } ;
} ;
[import and export statements omitted]

```

Example 2

The **clusterid** is omitted.

[autonomoussystem omitted]

```

routerid 192.0.2.1 ;
mpbgp on {
    group type internal peeras 65534 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
# Routes received from clients will have the clusterid of 192.0.2.1
# added to their BGP attributes.
    group type internal peeras 65534 reflector-client {
        peer 192.0.2.10 ;
        peer 192.0.2.11 ;
        peer 192.0.2.12 ;
        peer 192.0.2.13 ;
    } ;
} ;
[import and export statements omitted]

```

See Also

mpbgp on page 482

“Route Reflection” on page 71 in *Configuring GateD*

comm

Name

comm - specifies the community attributes added to routes sent to routers in a peer group

Syntax

comm {*community_values*}

Parameters

community_values include:

comm-hex *hex-number hex-number* - This parameter specifies any arbitrary community that is the concatenation of the two 16-bit numbers specified.

comm-split *autonomous_system community-id* - This parameter specifies a community that is the concatenation of the AS number ASN and the arbitrary 16-bit number.

community no-export - Specifies the well-known community NO_EXPORT as defined in RFC 1997. Routes tagged with this community are not to be exported outside of a confederation boundary.

community no-advertise - Specifies the well-known community NO_ADVERTISE as defined in RFC 1997. Routes tagged with this community are not to be advertised to any other peers.

community no-export-subconfed - Specifies the well-known community NO_EXPORT_SUBCONFED as defined in RFC 1997. Routes tagged with this community are not to be advertised to external peers, even if they are within the same confederation.

Description

comm specifies the community attributes added to routes sent to routers in a peer group. Communities may also be manipulated on import and export of routes from peers.

See "BGP Communities" on page 77 in *Configuring GateD* for more information.

Default

none

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

Examples

Example 1

Add the no-export community to a group.

[*autonomoussystem* and *routerid* omitted]

```
mpbgp on {
    group type external peeras 65534
        comm { community no-export ; } {
            peer 192.0.2.1;
        } ;
} ;
[import and export statements omitted]
```

Example 2

Add the community 64512:100 to a group.

[autonomoussystem and routerid omitted]

```
mpbgp on {
    group type external peeras 65534
        comm { comm-split 64512 100 ; } {
            peer 192.0.2.1;
        } ;
} ;
[import and export statements omitted]
```

Example 3

Add the community 64512:100 and community no-export-subconfed to a group.

[autonomoussystem and routerid omitted]

```
mpbgp on {
    group type external peeras 65534
        comm { comm-split 64512 100 ; community no-export-subconfed ; } {
            peer 192.0.2.1;
        } ;
} ;
[import and export statements omitted]
```

See Also

"BGP Communities" on page 77 in *Configuring GateD*

`import` on page 605

`export` on page 623

confed

Name

`confed` - marks a BGP group as being part of a BGP confederation

Syntax

`confed`

Parameters

none

Description

`confed` configures a BGP group to be part of a BGP confederation. The `confed-id` keyword must have been previously specified. When the `confed` keyword is present on a BGP group statement, GateD will treat all members of that group as confederation peers. Additionally, BGP will use the configured `autonomoussystem` number in its peering session for its AS number instead of `confed-id`.

When sending UPDATES to confederation peers, the AS_PATH is modified using AS_CONFED_SET and AS_CONFED_SEQUENCES instead of the normal CONFED_SET and CONFED_SEQUENCE. Additionally, the restriction against propagating MED and LOCAL_PREF values is relaxed and these values are propagated to confederation external peers.

Default

off

Context

`bgp group` statement

Examples

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;
mpbgp on {
    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 mpbgp 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
# propagate everything from our confederation.
export bgp as 200 {
    proto bgp as 64512 {
        all ;
    } ;
    proto bgp as 65000 {
        all ;
    } ;
} ;
```

See also

RFC 3065 at <http://www.ietf.org/rfc/rfc3065.txt>

defaultmetric

Name

defaultmetric - defines the metric (MED) used when advertising routes via MPBGP

Syntax

```
defaultmetric metric ;
```

Parameters

metric

Description

defaultmetric defines the metric (MED) used when advertising routes via MPBGP. If not specified, no metric is propagated. This metric may be overridden by a metric specified on the **peer** or **group** statements or in export policy.

Default

none

Context

mpbgp statement

Examples

```
[autonomoussystem and routerid omitted]
mpbgp on {
    defaultmetric 100 ;
    group type external peeras 64512 {
# Metric of 100 is sent to this peer by default.
        peer 192.0.2.2 ;
# Metric of 150 is sent to this peer.
        peer 192.0.2.3 metricout 150 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

metricout on page 480

discard-nonprefixed-confederations

Name

discard-nonprefixed-confederations - discards malformed AS_PATHs containing non-prefixed confederation segments

Syntax

discard-nonprefixed-confederations

Parameters

none

Description

discard-nonprefixed-confederations causes MPBGP to discard AS_PATHs that are received from BGP peers where there are BGP Confederation AS_PATH segments (AS_CONFED_SEQUENCE, AS_CONFED_SET) occurring anywhere other than at the left hand side of the AS_PATH. This feature is useful because of “buggy” routers on the Internet that will illegally advertise AS_PATHs containing confederation segments outside of a confederation boundary. If these routes propagate beyond the confederation boundary edge, they will cause the peering session of any router that does not accept confederation segments from non-confederation peers to drop the peering session and thus disrupt service.

This option will cause routes containing non-prefixed confederation segments to be logged and discarded.

Default

not enabled

Context

mpbgp statement

Examples

```
routerid 192.0.0.1;
autonomoussystem 64512;

mpbgp on {
    discard-nonprefixed-confederations;

    group type external peeras 65534 {
        allow {
            all;
        };
    };
};
```

```
};
```

```
[ static, import, and export clauses omitted ]
```

See Also:

“Chapter 22 Multi-Protocol - Border Gateway Protocol (MPBGP)” on page 111 of *Configuring GateD*

gateway

Name

gateway - instructs GateD to use a form of multihop External Border Gateway Protocol (EBGP)

Syntax

```
gateway gateway_ip_address ;
```

Parameters

gateway_ip_address - A gateway is an intermediate destination by which packets are delivered to their ultimate destination. A gateway is the IP address of the gateway host.

Description

gateway instructs GateD to use a form of multihop EBGP. If a network is not shared with this group, **gateway** specifies a router on an attached network to be used as the next hop router for routes received from this peer. The **gateway** parameter may also be used to specify a next hop for groups that are on shared networks. For example, you might use **gateway** to ensure that third-party next hops are never accepted from a given group by specifying that group's address as its own gateway. The **gateway** specified must have consistent routing information to prevent routing loops. **gateway** is not needed in most cases.

Default

none

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

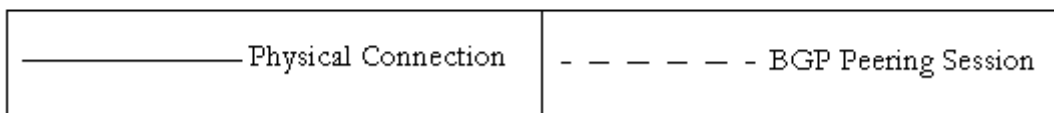
mpbgp peer statement

Examples

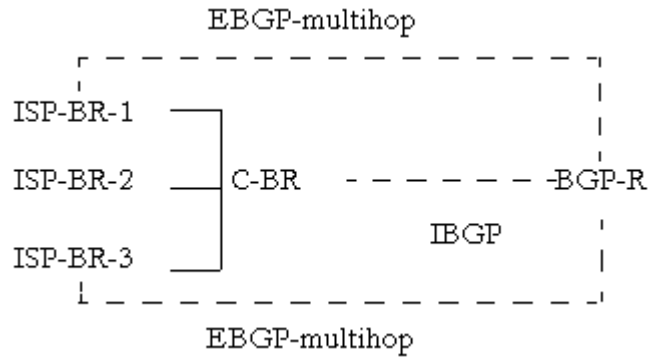
Consider the following example:

THE ROUTE SIEVE

Key:



A company has three Internet feeds. The customer border router has direct connections to the ISP border routers. However, the company's router does not have enough resources to hold three full views of the Internet. The company uses a GateD workstation running EBGP multihop to establish peering sessions with the ISP border routers and uses IBGP to send the selected routes to the EBGP-multihop company border router.



Router	AS	Router ID/Interface
ISP-BR-1	65501	192.168.1.1 -- DS3 to C-BR
ISP-BR-2	65502	10.0.0.1 -- DS3 to C-BR
ISP-BR-3	65503	172.16.0.1 -- DS3 to C-BR
C-BR	64512	192.0.2.1 -- Gig Ethernet to BGP-R
BGP-R	64512	192.0.2.2 -- Gig Ethernet to C-BR

Example 1

```

autonomoussystem 64512 ;
routerid 192.0.2.1 ;
mpbgp on {
    group type external peeras 65501 gateway 192.168.1.1 {
        peer 192.168.1.1;
    } ;
    group type external peeras 65502 gateway 10.0.0.1 {
        peer 10.0.0.1 ;
    } ;
    group type external peeras 65503 gateway 172.16.0.1 {
        peer 172.16.0.1 ;
    } ;
} ;
  
```

```
        group type internal peeras 64512 {  
            peer 192.0.2.1  
        } ;  
    } ;
```

[import and export statements omitted]

Example 2

```
autonomoussystem 64512 ;  
routerid 192.0.2.1 ;  
mpbgp on {  
    group type external peeras 65501 {  
        peer 192.168.1.1 gateway 192.168.1.1 ;  
    } ;  
    group type external peeras 65502 {  
        peer 10.0.0.1 gateway 10.0.0.1 ;  
    } ;  
    group type external peeras 65503 {  
        peer 172.16.0.1 gateway 172.16.0.1 ;  
    } ;  
    group type internal peeras 64512 {  
        peer 192.0.2.1  
    } ;  
}
```

[import and export statements omitted]

See Also

"Third Party Routing" on page 70 in *Configuring GateD*

group type

Name

group type - specifies an MPBGP group as internal, external or routing

Syntax

```
group type external peer as autonomous_system
group type internal peer as autonomous_system
group type routing peer as autonomous_system proto protocol
```

Parameters

autonomous_system - a number between 1 and 65535 inclusive, specifying a set of routers under a single technical administration and assigned by the Internet Assigned Numbers Authority

protocol - name of the protocol to be used to resolve MPBGP route next hops, including **static**, **rip**, **ospf**, and **isis**

Groups also have the following parameters, which are described throughout the BGP section of this manual:

```
ascount count
caps cap_list
comm community_list
confed
gateway host
holdtime time
ignorefirstashop
keep ( all | none )
keepalivesalways
localtcp local_address
localas autonomous_system
localv4addr ipv4_address
localv6addr ipv6_address
logupdown
med
metricout metric
nexthopself
noagggregatorid
nogendefault
nov4asloop
outdelay time
passive
preference group preference
preference2 group preference2
recvbuffer buffer_size
reflector-client [ no-client-reflect ]
remotev4addr ipv4_address
remotev6addr ipv6_address
routetopeer
sendbuffer buffer_size
```

```
setpref metric
showwarnings
traceoptions trace_options
ttl ttl
```

Description

Within a group, MPBGP peers may be configured in one of two ways. They may be implicitly configured with the **allow** statement or explicitly configured with a **peer** statement.

In **group type external**, full policy checking is applied to all incoming and outgoing advertisements. The external peers must be directly reachable through one of the machine's local interfaces. If they are not, BGP may be "multi-hopped" through the use of the **gateway** statement. The next hop transmitted is computed with respect to the shared interface.

The **group type internal** specifies an internal group operating where there is no IP-level IGP, for example, an SMDS network or MILNET. If the peer is not directly connected, the **route-to-peer** and **ttl** statements may be used to "multihop" the IBGP session. Import and export policy may be applied to group advertisements. Routes received from external MPBGP peers are by default readvertised with the received metric.

The **group type routing** propagates external routes between routers that are not directly connected. **group type routing** also computes immediate next hops for those external routes by using the MPBGP next hop that arrived with the route as a forwarding address to be resolved via an internal protocol's routing information. In essence, internal MPBGP is used to carry AS external routes, and the IGP is expected to carry only AS internal routes. The latter is used to find immediate next hops for the former. **proto protocol** names the interior protocol to be used to resolve MPBGP route next hops, and may be the name of any IGP in the configuration, including static. By default, the next hop in MPBGP routes advertised to **group type routing** peers will be set to the local address on the BGP connection to those peers because it is assumed that a route to this address will be propagated via the IGP. The **interface list** can optionally provide a list of interfaces whose routes are carried via the IGP for which third-party next hops may be used instead.

localtcp, **outdelay**, and **metricout** must be set in the **group** clause, not on a per-peer basis, for the group types **internal** and **routing**. If these options are set on the **peer** sub-clause, they must equal the values set on the corresponding **group** clause.

Default

none

Context

mpbgp statement

Examples

```
autonomoussystem 65531
mpbgp on {
    group type external peeras 65534 {
        peer 192.0.2.2 ;
    }
}
```

```
    } ;  
    group type internal peeras 65531 {  
        peer 192.0.2.3 ;  
    } ;  
    group type routing peeras 65531 proto ospf {  
        peer 192.0.2.4 ;  
    } ;  
} ;
```

See Also

[mpbgp](#) on page 482

holdtime

Name

holdtime - specifies the MPBGP hold time value, in seconds, to use when negotiating a peering session

Syntax

holdtime *time*

Parameters

time - time in seconds, an integer that is at least 3

Description

holdtime specifies the MPBGP hold time value, in seconds, to use when negotiating a peering session with this group. If GateD does not receive a keepalive, update, or notification message within the period specified in the hold time field of the MPBGP Open message, then the MPBGP connection will be closed. The value must be at least 3.

The negotiated holdtime value is the lesser of the values sent in the exchanged MPBGP Open messages.

Default

holdtime 180 ;

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

holdtime set in the **group** statement

[autonomoussystem and routerid omitted]

mpbgp on {

group type external **peeras** 64512 **holdtime** 30 {

peer 192.0.2.1 ; # Holdtime is 30 seconds

peer 192.0.2.2 ; # Holdtime is 30 seconds

 } ;

} ;

[import and export statements omitted]

Example 2

`holdtime` set in the `group` and `peer` statements

[`autonomous` system and `routerid` omitted]

```
mpbgp on {  
    group type external peer as 64512 holdtime 90 {  
        peer 192.0.2.1 ; # Holdtime is 90 seconds  
        peer 192.0.2.2 holdtime 30 ; # Holdtime is 30 seconds  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

`keepalivesalways` on page 467

ignorefirstashop

Name

`ignorefirstashop` - directs GateD to keep route servers' routes

Syntax

`ignorefirstashop ;`

Parameters

none

Description

Some routers, known as "route servers," are capable of propagating routes without appending their own autonomous system (AS) number to the AS Path. By default, GateD will drop such routes. Specifying `ignorefirstashop` on the `group` clause allows GateD to keep these routes. `ignorefirstashop` should be used only if there is no doubt that these peers are route servers and not normal routers.

Default

disabled

Context

`mpbgp group type external` statement
`mpbgp peer` statement for external groups

Examples

Example 1

`ignorefirstashop` set in `group` statement
[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 ignorefirstashop {  
        peer 192.0.2.1 ;  
        peer 192.0.2.2 ;  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

`ignorefirstashop` set in `peer` statement

[`autonomous system` and `routerid` omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ignorefirstashop ;  
        peer 192.0.2.2 ;  
    } ;  
} ;
```

[`import` and `export` statements omitted]

ignore-nonprefixed-confederations

Name

ignore-nonprefixed-confederations - ignores malformed AS_PATHs containing non-prefixed confederation segments

Syntax

```
ignore-nonprefixed-confederations ;
```

Parameters

none

Description

ignore-nonprefixed-confederations causes BGP to ignore AS_PATHs that are received from BGP peers where there are BGP Confederation AS_PATH segments (AS_CONFED_SEQUENCE, AS_CONFED_SET) occurring anywhere other than at the left-hand side of the AS_PATH. This feature is useful due to "buggy" routers on the Internet that will illegally advertise AS_PATHs containing confederation segments outside of a confederation boundary. If these routes propagate beyond the confederation boundary edge, they will cause the peering session of any router that does not accept confederation segments from non-confederation peers to drop the peering session and thus disrupt service. This option will cause routes containing non-prefixed confederation segments to be logged and stored in the RIB with a preference of -1. The GII command "**show mpbgp bad-as-path**" will then show all of these routes.

Default

not enabled

Context

mpbgp statement

Examples

```
routerid 192.0.0.1;  
autonomoussystem 64512;  
  
bgp on {  
    ignore-nonprefixed-confederations;  
  
    group type external peeras 65534 {  
        allow {  
            all;  
        };
```

```
};  
};  
[ static, import, and export clauses omitted ]
```

See Also

"Chapter 22 Multi-Protocol - Border Gateway Protocol (MPBGP)" on page 111 of *Configuring GateD*

discard-nonprefixed-confederations on page 450

RFC 3065 - Autonomous System Confederations for BGP

interface

See **interface** on page 28

keep

Name

keep - specifies whether to keep routes containing a router's own autonomous system (AS) number

Syntax

```
keep ( all | none );
```

Parameters

all or none

Description

keep all retains routes learned from a peer even if the routes' AS paths contain the router's own AS number.

keep none causes GateD to disregard routes containing the router's own AS number.

Default

```
keep none ;
```

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

keep all set in group statement

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 keep all {  
        peer 192.0.2.1 ; # keeps all  
        peer 192.0.2.2 ; # keeps all  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

keep all set in peer statement

[autonomous system and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 keep all ;  
        peer 192.0.2.2 ; # keeps none  
    } ;  
} ;  
[import and export statements omitted]
```

keepalivesalways

Name

keepalivesalways - causes GateD to always send keepalives

Syntax

```
keepalivesalways ;
```

Parameters

none

Description

keepalivesalways causes GateD to always send keepalives, even when an **update** could have correctly substituted for one. **keepalivesalways** allows interoperability with routers that do not completely obey the protocol specifications on this point.

Default

disabled

Context

```
mpbgp group type external statement  
mpbgp group type internal statement  
mpbgp group type routing statement  
mpbgp peer statement
```

Examples

Example 1

keepalivesalways set in group statement

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 keepalivesalways {  
        peer 192.0.2.1 ; # keepalivesalways  
        peer 192.0.2.2 ; # keepalivesalways  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

keepalivesalways set in peer statement

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 keepalivesalways ;  
        peer 192.0.2.2 ; # Normal behavior  
    } ;  
} ;  
[import and export statements omitted]
```

localas

Name

localas - identifies the autonomous system that GateD is representing to this group of peers

Syntax

```
localas autonomous_system ;
```

Parameters

autonomous_system - a number between 1 and 65535 inclusive, specifying a set of routers under a single technical administration and assigned by the Internet Assigned Numbers Authority

Description

localas identifies the autonomous system that GateD is representing to this group of peers. The default is that which has been set globally in the **autonomoussystem** statement.

Default

inherited from the global **autonomoussystem**

Context

mpbgp group type external statement

Examples

```
[routerid omitted]
autonomoussystem 64512
mpbgp on {
    group type external peeras 65000 { # use as 64512 (inherited)
        peer 192.0.2.1 ; # keepalivealways
    } ;
    group type external peeras 65001 localas 65534 {
        peer 192.0.2.2 ; # keepalivealways
    } ;
} ;
[import and export statements omitted]
```

localtcp

Name

localtcp - specifies the address to be used on the local end of the TCP connection with the group

Syntax

```
localtcp local_address ;
```

Parameters

local_address - 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 or in an IPv4 dotted-quad format (a.b.c.d) or IPv6 format (a : b : c : d :: e).

Description

localtcp specifies the IP address to be used on the local end of the TCP connection with the peer. For external peers, the local address must be on an interface that is shared with the peer or with the peer's gateway when **gateway** is used. A session with an external peer will be opened only when an interface with the appropriate local address (through which the peer or gateway address is directly reachable) is operating. For **internal** and **routing** peers, a peer session will be maintained when any interface with the specified local address is operating. In any case, an incoming connection will be recognized as a match for a configured peer only if it is addressed to the configured local address.

For group types **internal** and **routing**, set this **localtcp** on the **group** clause.

Default

the IP address of a shared interface

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 localtcp 192.168.1.1 {  
        peer 192.0.2.1 ;  
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

```
mpbgp on {  
    group type external peeras 64513 caps {v6u} localv4addr 192.0.2.2  
    localtcp fec0:0:0:ff::d {  
        peer fec0:0:0:ff::e ;  
    } ;  
} ;
```

localv4addr

Name

localv4addr - specifies an IPv4 address to send as a next hop when advertising IPv4 NLRI to an IPv6 peer.

Syntax

localv4addr - *ipv4 address* ;

Parameters

ipv4 address

Description

BGP peering sessions can be established over IPv4 or IPv6, but NLRI of both address families may be advertised to a given peer. When peering over IPv6, the configured address of the peer is used to determine acceptable next hops to advertise to that peer. However, GateD has no knowledge of the peer's IPv4 configuration. **localv4addr** specifies an IPv4 address for GateD to advertise as a next hop for IPv4 NLRI advertised to IPv6 peers. **localv4addr** must be configured for all IPv6 peers unless an appropriate address can be determined from the configuration of **remotev4addr**.

Default

none (A parse error will result if not configured and it is required.)

Context

mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement

Examples

Example 1

```
mpbgp on {  
    group type external peeras 64512 caps {v4u v6u} localv4addr  
        192.0.2.1 {  
            peer fec0:0:0:ff::d;  
        };  
};
```

Example 2

```
mpbgp on {
```

```
group type external peeras 64512 caps {v4u v6u} localv4addr
  192.0.2.1 {
    peer fec0:0:0:ff::d;
    # override group definition of localv4addr
    peer fec0:0:0:ff::e localv4addr 192.0.2.10;
  };
};
```

See Also

`localv6addr` on page 474
`remotev4addr` on page 500
`remotev6addr` on page 502

localv6addr

Name

localv6addr - specifies an IPv6 address to send as a next hop when advertising IPv6 NLRI to an IPv4 peer

Syntax

```
localv6addr - ipv6_address ;
```

Parameters

ipv6_address

Description

BGP peering sessions can be established over IPv4 or IPv6, but NLRI of both address families may be advertised to a given peer. When peering over IPv4, the configured address of the peer is used to determine acceptable next hops to advertise to that peer. However, GateD has no knowledge of the peer's IPv6 configuration. **localv6addr** specifies an IPv6 address for GateD to advertise as a next hop for IPv6 NLRI advertised to IPv4 peers. **localv6addr** must be configured for all IPv6 peers unless an appropriate address can be determined from the configuration of **remotev6addr**.

Default

none (A parse error will result if not configured and it is required.)

Context

```
mpbgp group type external statement  
mpbgp group type internal statement  
mpbgp group type routing statement  
mpbgp peer statement
```

Examples

Example 1

```
mpbgp on {  
    group type external peer as 64512 caps {v4u v6u} localv6addr  
        fec0:0:0:ff::d {  
            peer 192.0.2.1;  
        }  
};
```

Example 2

```
mpbgp on {
```

```
group type external peeras 64512 caps {v4u v6u} localv6addr
    fec0:0:0:ff::d {
        peer 192.0.2.1;
        # override group definition of localv6addr
        peer 192.0.2.2 localv6addr fec0:0:0:ff::e;
    };
};
```

See Also

`localv4addr` on page 472
`remotev4addr` on page 500
`remotev6addr` on page 502

logupdown

Name

logupdown - causes a message to be logged via the syslog mechanism whenever a BGP group enters or leaves the Established state

Syntax

```
logupdown ;
```

Parameters

none

Description

logupdown causes a message to be logged via the syslog mechanism whenever a BGP group enters or leaves the Established state.

Default

disabled

Context

```
mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement
```

Examples

Example 1

Enable **logupdown** in the group.

[autonomoussystem and routerid omitted]

```
mpbgp on {
    group type external peeras 64512 logupdown {
        peer 192.0.2.1 ; # logupdown enabled
        peer 192.0.2.2 ; # logupdown enabled
    } ;
} ;
```

[import and export statements omitted]

Example 2

Enable **logupdown** in the peer.

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 logupdown ; # logupdown enabled  
        peer 192.0.2.2 ;           # logupdown disabled  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

`traceoptions` on page 508

med

Name

med - allows MEDs to be used in routing computations

Syntax

med ;

Parameters

none

Description

By default, any MED (*Multi_Exit_Disc*) received on a BGP connection is ignored. If MEDs are to be used in routing computations, the **med** option must be specified on the **group** or **peer** clauses. By default, MEDs are not sent on external connections. To send MEDs, use the **metric** option of the **export** statement or the **metricout** peer/group parameter.

When two routes to the same destination are received from different peers within the same **peer-as**, they could have different MEDs. When choosing between these routes, assuming that nothing else makes one preferable to the other (such as configured policy), the values of the differing MEDs are used to choose which route to use. In this comparison, the route with the lowest MED is preferred. Routes without MEDs are treated as having the best possible MED. MEDs are not propagated to internal peers unless **med** is enabled.

Default

disabled

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

Enable **med** processing on the **group** statement.

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 med {  
        peer 192.0.2.1 ; # med comparison is enabled  
        peer 192.0.2.2 ; # med comparison is enabled  
    }  
}
```

```
    } ;  
} ;  
[import and export statements omitted]
```

Example 2

Enable `med` processing on the `peer` statement.

[`autonomous system` and `routerid` omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 med ; # med comparison is enabled  
        peer 192.0.2.2 ;    # med comparison is disabled  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

“Preferences and Route Selection” on page 11 in *Configuring GateD*

metricout

Name

metricout - causes BGP to send a Multi-Exit Discriminator (MED) when routes are advertised to peers

Syntax

```
metricout metric ;
```

Parameters

metric

Description

The **metricout** statement causes a BGP MED to be set on routes when they are advertised to peers. The metric hierarchy is as follows, starting from the most preferred:

1. the metric specified by export policy
2. peer-level **metricout**
3. group-level **metricout**
4. **defaultmetric**

For group types **internal** and **routing**, set **metricout** on the **group** clause instead of on the **peer** subclause (MED needs to be common among all peers in an internal group or looping may occur).

Default

no metric, or, if set, **defaultmetric**

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

Set a **metricout** of 50 on the **group**.

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 metricout 50 {  
        peer 192.0.2.1 ; # Send MED of 50  
        peer 192.0.2.2 ; # Send MED of 50  
    }  
}
```

```

        peer 192.0.2.3 ; # Send MED of 50
    } ;
} ;

```

[import and export statements omitted]

Example 2

Set a `metricout` of 50 on a specific `peer`.

[autonomoussystem and routerid omitted]

```

mpbgp on {
    group type external peeras 64512 {
        peer 192.0.2.1 metricout 50 ; # Send MED of 50
        peer 192.0.2.2 ;                # Send no MED
        peer 192.0.2.3 ;                # Send no MED
    } ;
} ;

```

[import and export statements omitted]

Example 3:

Set a `metricout` of 50 on a specific `peer`.

[autonomoussystem and routerid omitted]

```

defaultmetric 100;
mpbgp on {
    group type external peeras 64512 {
        peer 192.0.2.1 metricout 50 ; # Send MED of 50
        peer 192.0.2.2 ;                # Send MED of 100
        peer 192.0.2.3 ;                # Send MED of 100
    } ;
} ;

```

[import and export statements omitted]

See Also

`defaultmetric` on page 449

"Preferences and Route Selection" on page 11 in *Configuring GateD*

mpbgp

Name

mpbgp - enables or disables MPBGP

Syntax

mpbgp [**on** | **off**] {*mpbgp_parameters*}

Parameters

mpbgp_parameters - includes all the parameters in this section

Description

mpbgp enables or disables MPBGP. *mpbgp_parameters* includes all the parameters in this section.

Default

mpbgp off ;

Context

global

Examples

```
mpbgp on {  
    group type external peeras 64512 ignorefirstashop {  
        peer 192.0.2.1 ;  
        peer 192.0.2.2 ;  
    } ;  
} ;
```

See Also

“Multi-Protocol - Border Gateway Protocol” on page 111 of *Configuring GateD*

nexthopself

Name

nexthopself - sets this group or peer's next hop to the router's own address on advertisement

Syntax

nexthopself

Parameters

none

Description

nexthopself sets this group's next hop to the router's own address even if it would normally be possible to send a third-party next hop.

nexthopself can cause inefficient routes to be followed. It might be needed in some cases to deal with improperly bridged interconnect media (in cases where the routers on the "shared" medium do not really have full connectivity to each other) or when political situations cause broken links.

nexthopself can be used only for external groups.

Default

disabled

Context

mpbgp group type external statement

mpbgp peer statement (in external groups)

Examples

Consider the following topology:

```
R1----R2
      |
      R3
```

Three routers that have interfaces all in the same subnet are in a partially meshed ATM network.

- R1 and R2 have a Permanent Virtual Circuit (PVC.)
- R2 and R3 have a PVC.
- R1 and R3 do not have any direct connectivity.
- R1, R2, and R3 share a common subnet.

Because all routers are in the same subnet, normally, routes exchanged between R1, R2, and R3 would all use third-party routing. This would result in routes exchanged between R1 and R3 to result in a blackhole because R1 and R3 do not have a valid physical connection.

When R2 exchanges routes with R1 and R3, it should use `nexthopself` to disable the third-party routing.

Configuration for R2:

```
mpbgp on {  
    group type external peeras 64512 nexthopself {  
        peer 192.0.2.1 ; # R1  
    } ;  
    group type external peeras 64513 nexthopself {  
        peer 192.0.2.2 ; # R2  
    } ;  
} ;
```

[import and export statements omitted]

This could also be written as:

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 nexthopself ; # R1  
    } ;  
    group type external peeras 64513 {  
        peer 192.0.2.2 nexthopself ; # R2  
    } ;  
} ;
```

See Also

“Third Party Route Advertisement” on page 70 in *Configuring GateD*

noaggregatorid

Name

`noaggregatorid` - causes GateD to specify the `routerid` in the aggregator attribute as 0

Syntax

`noaggregatorid ;`

Parameters

none

Description

`noaggregatorid` causes GateD to specify the `routerid` in the aggregator attribute as 0 (instead of the `routerid` of the router) in order to prevent different routers in an autonomous system (AS) from creating aggregate routes with different AS Paths.

Default

disabled

Context

`mpbgp group type external` statement
`mpbgp group type internal` statement
`mpbgp group type routing` statement
`mpbgp peer` statement

Examples

```
[autonomoussystem and routerid omitted]
mpbgp on {
    group type external peeras 64512 noaggregatorid {
        peer 192.0.2.1 ;
    } ;
# Any aggregates will have a routerid of 0.0.0.0.
    aggregate 10.0.0.0 masklen 9 {
        proto bgp {
            10.0.2.0 masklen 24 ;
            10.0.4.0 masklen 24 ;
        } ;
    } ;
} ;
[import and export statements omitted]
```

See Also

“Route Aggregation” on page 155 in *Configuring GateD*

nogendefault

Name

nogendefault - prevents GateD from generating a default route when BGP receives a valid update from its peer

Syntax

```
nogendefault ;
```

Parameters

none

Description

nogendefault prevents GateD from generating a default route when MPBGP receives a valid update from its peer. The default route is generated only when the **gendefault** option is enabled.

Default

disabled

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {
```

```
    group type external peeras 64512 nogendefault {
```

```
        peer 192.0.2.1 ;
```

```
    } ;
```

```
} ;
```

[import and export statements omitted]

See Also

"Options Statements" on page 21 in *Configuring GateD*

nov4asloop

Name

nov4asloop - prevents routes with looped autonomous system (AS) Paths from being advertised to version 4 external peers

Note: **nov4asloop** is deprecated.

Syntax

```
nov4asloop ;
```

Parameters

none

Description

nov4asloop prevents routes with looped AS paths from being advertised to version 4 external peers. Use **nov4asloop** to avoid advertising routes to peers that would incorrectly forward the routes on to version 3 peers.

In this context, "looped" refers to "AS Path stuffing" where a given AS is inserted multiple times in an AS Path.

Default

disabled

Context

```
mpbgp group type external statement  
mpbgp group type internal statement  
mpbgp group type routing statement  
mpbgp peer statement
```

Examples

```
[autonomoussystem and routerid omitted]  
mpbgp on {  
    group type external peeras 64512 nov4asloop {  
        peer 192.0.2.1 ;  
    };  
};  
[import and export statements omitted]
```

See Also

"BGP Communities" on page 133 of *Configuring GateD*

open-on-accept

Name

open-on-accept - causes GateD to send a BGP Open message immediately after the transport (TCP) connection has completed

Syntax

open-on-accept ;

Parameters

None

Description

When GateD receives an incoming BGP peering session, it will delay sending the BGP Open message for a short time after the transport connection (TCP) has completed. This is a violation of the BGP Finite State Machine, but is used to allow the calling peer to send its Open message first. This is used to determine if a peering session matches a given **allow** clause. When the **open-on-accept** keyword is present, GateD will immediately send the Open message when the TCP connection has completed for configured peers. If the peer is unconfigured (is matched by an **allow** clause, but not a **peer** keyword), or is **passive**, GateD will continue to wait for the Open message from the remote peer before sending its own BGP Open message.

Default

Off

Context

mpbgp statement

Examples

```
autonomoussystem 64512;  
mpbgp yes {  
    open-on-accept ;  
    group type external peeras 65534 {  
        peer 192.168.1.1;  
    };  
};
```

See Also

"Chapter 22 Multi-Protocol - Border Gateway Protocol (MPBGP)" on page 111 of *Configuring GateD*

outdelay

Name

`outdelay` - damps route fluctuations

Syntax

`outdelay time ;`

Parameters

time - time in seconds

Description

`outdelay` damps route fluctuations. The `outdelay time` is the amount of time a route must be present in the GateD routing database before it is exported to MPBGP. `outdelay` is intended only for use with external peers.

Note: Weighted Route Damping is better suited for improving overall network stability. The use of this option may delay route convergence for well-behaved routers.

Default

`outdelay 0 ;` (This feature is disabled.)

Context

`mpbgp group type external` statement

`mpbgp peer` statement for external groups

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {
    group type external peeras 64512 outdelay 10 {
        peer 192.0.2.1 ;
    };
    group type external peeras 64513 {
        peer 192.0.2.2 outdelay 15 ;
        peer 192.0.2.3 ; # no outdelay
    };
} ;
```

[import and export statements omitted]

See Also

"Route Flap Damping" on page 163 in *Configuring GateD*

passive

Name

passive - prevents GateD from ever trying to open a BGP connection with peers in this group

Syntax

```
passive ;
```

Parameters

none

Description

passive prevents GateD from ever trying to open a BGP connection with peers in this group. Instead, the router will wait for the peer to initiate a connection. **passive** was introduced to handle a problem in BGP3 and earlier in which two peers might both attempt to initiate a connection at the same time. This problem is fixed in the BGP4 protocol, so the **passive** option is not needed with BGP4 sessions.

Note: If it is applied to both sides of a peering session, **passive** will prevent the session from ever being established. For this reason, and because it is generally not needed, the use of **passive** is discouraged.

Default

disabled

Context

```
mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement
```

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {
    group type external peeras 64512 passive {
        peer 192.0.2.1 ;
    } ;
} ;
```

[import and export statements omitted]

peer

Name

peer - configures an individual peer

Syntax

```
peer host [ peer_options ] ;
```

Parameters

host - the IPv6 or IPv4 address of the host machine

Peer options are described throughout the BGP section and include:

```
ascount count
gateway gateway
holdtime time
ignorefirstashop
keep ( all | none )
keepalivesalways
localas autonomous_system
localtcp local_address
localv4addr ipv4_address
localv6addr ipv6_address
logupdown
med
metricout metric
nexthopself
noagggregatorid
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
```

Description

peer configures an individual peer. Each peer inherits all parameters specified on a **group** clause as defaults. Many defaults may be overridden by parameters explicitly specified on the **peer** subclause. Within each **group** clause, individual peers can be specified, or a group of potential peers can be specified using **allow**. Use **allow** to specify a set of address

masks. If GateD receives an MPBGP connection request from any address in the set specified, it will accept it and set up a peer relationship.

Default

not applicable

Context

`mpbgp` statement

Examples

Example 1

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ;  
        peer 10.0.0.1 passive ;  
        peer 192.168.1.1 nexthopself ;  
    } ;  
} ;
```

[import and export statements omitted]

Example 2

```
mpbgp on {  
    group type external peeras 64513 caps {v6u} localv4addr 192.168.1.2 {  
        peer } ffe:0:0:ff::e ;  
    } ;  
} ;
```

preference

Name

preference - specifies how active routes that are learned from MPBGP (compared to other protocols) will be selected

Syntax

preference *mpbgppreference*

Parameters

mpbgppreference - an assigned integer between 0 (directly connected) and 255 (for EGP)

Description

preference specifies how active routes that are learned from MPBGP (compared to other protocols) 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. This **preference** may be overridden by a **preference** specified on the **group** or **peer** statements or by import policy.

Default

preference 170 ;

Context

mpbgp statement

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

Example 1

The following example sets the global MPBGP preference to 140, which makes MPBGP routes better than OSPF AS-external routes.

[autonomoussystem and routerid omitted]

```
mpbgp on {
  preference 140 ;
    group type external peeras 64512 {
      peer 192.0.2.1 ;
    } ;
}
```

[import and export statements omitted]

Example 2

The following example sets the preference for a specific MPBGP group to be better than routes from another MPBGP group. This ensures that, for a given prefix, these routes are always preferred.

[autonomous system and routerid omitted]

```
mpbgp on {  
  # These routes are always preferred compared with  
  # other mpbgp routes  
  group type external peeras 64512 preference 165 {  
    peer 192.0.2.1 ;  
  } ;  
  group type external peeras 65000 { # default preference of 170  
    peer 192.0.2.2 ;  
  } ;  
} ;
```

[import and export statements omitted]

Example 3

Policy for all

[autonomous system and routerid omitted]

```
mpbgp on {  
  group type external peeras 64512 {  
    peer 192.0.2.1 ;  
  } ;  
} ;  
  
import proto bgp as 64512 {  
  # We will prefer this route over almost anything other than  
  # a directly attached interface.  
  10.0.0.0 masklen 24 preference 5 ;  
  all ; # default 170  
} ;
```

[export statement omitted]

See Also

“Preferences and Route Selection” on page 11 in *Configuring GateD*

preference2

Name

preference2 - breaks a **preference** tie between groups

Syntax

```
preference2 grouppreference2 ;
```

Parameters

*group***preference2** - an integer between 0 and 255

Description

preference2 breaks a **preference** tie between groups. Preferences are the first criteria of comparison for route selection.

Default

```
preference2 0 ;
```

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    # Prefer these routes.  
    group type external peeras 64512 preference 5 {  
        peer 192.0.2.1 ;  
    } ;  
    group type external peeras 65000 { # default preference 170  
    # Routes from this peer are preferred over 192.0.2.3.  
        peer 192.0.2.2 preference2 10 ;  
        peer 192.0.2.3 preference2 20 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

“Preferences and Route Selection” on page 11 in *Configuring GateD*

recvbuffer

Name

recvbuffer - controls the amount of memory requested from the kernel for the receive buffer

Syntax

```
recvbuffer buffer_size ;
```

Parameters

buffer_size - an integer between 1 and 65535

Description

recvbuffer controls the amount of memory requested from the kernel for the receive buffer. The maximum supported is 65535 bytes, although many kernels have a lower limit. By default, GateD configures the maximum supported. **recvbuffer** is not needed on normally functioning systems.

Default

```
recvbuffer 65535 ;
```

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 recvbuffer 32768 {  
        peer 192.0.2.1 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

sendbuffer on page 505

reflector-client [no-client-reflect]

Name

reflector-client - specifies that GateD will act as a route reflector for this group
no-client-reflect - specifies that GateD will not act as an intra-group reflector and thus will not reflect routes back to peers within the same group. If used, this keyword must follow **reflector-client**.

Syntax

```
reflector-client [ no-client-reflect ] ;
```

Parameters

none

Description

reflector-client specifies that GateD will act as a route reflector for this group.
no-client-reflect specifies that GateD will not act as an intra-group reflector and thus will not reflect routes back to peers within the same group. This is used when client peers within a route-reflection group are fully meshed.

Default

none

Context

mpbgp group type (only internal or routing group types)

Examples

```
[autonomoussystem omitted]
routerid 192.0.2.1 ;
mpbgp on {
    clusterid 192.0.2.254 ;
    group type internal peeras 65534 {
        peer 192.0.2.2 ;
        peer 192.0.2.3 ;
    } ;
    # Routes received from clients will have the clusterid of
    # 192.0.2.254 added to their BGP attributes.
    group type internal peeras 65534 reflector-client {
        peer 192.0.2.10 ;
        peer 192.0.2.11 ;
        peer 192.0.2.12 ;
```

```
        peer 192.0.2.13 ;  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

“Route Reflection” on page 71 in *Configuring GateD*

remotev4addr

Name

remotev4addr - specifies a peer's IPv4 address to use as a next hop for routes learned from an IPv6 peer sending IPv4 NLRI to GateD. Also used to determine a value to use for **localv4addr**.

Syntax

remotev4addr - *ipv4_address* ;

Parameters

ipv4_address

Description

BGP peering sessions can be established over IPv4 or IPv6, but NLRI of both address families may be learned from a given peer. IBGP peers may advertise next hops on networks to which we are not directly connected. When the advertised routes have the same address family as the IBGP peering session, BGP can simply install the route with the next hop of the peer's address. In the case of an IPv6 IBGP peer advertising IPv4 NLRI, **remotev4addr** is used to specify a next hop for the routes.

remotev4addr can also be specified instead of **localv4addr**. If a local interface is configured on the same network as **remotev4addr**, this interface address will be used for **localv4addr** if it is unspecified.

Default

none

Context

mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement

Examples

```
mpbgp on {
    group type internal peeras 64512 {v4u v6u} remotev4addr
        192.0.1.1 {
            peer fec0:0:0:ff::d;
        };
    group type external peeras 64513 {v4u v6u} remotev4addr
        192.0.1.2 {
            peer fec0:0:0:ff::e;
        };
}
```

```
    };  
};
```

See Also

`localv4addr` on page 472

`localv6addr` on page 474

`remotev6addr` on page 502

remotev6addr

Name

remotev6addr - specifies a peer's IPv6 address to use as a next hop for routes learned from an IPv4 peer sending IPv6 NLRI to GateD. Also used to determine a value to use for **localv6addr**.

Syntax

remotev6addr - *ipv6_address* ;

Parameters

ipv6_address

Description

BGP peering sessions can be established over IPv4 or IPv6, but NLRI of both address families may be learned from a given peer. IBGP peers may advertise next hops on networks to which we are not directly connected. When the advertised routes have the same address family as the IBGP peering session, BGP can simply install the route with the next hop of the peer's address. In the case of an IPv4 IBGP peer advertising IPv6 NLRI, **remotev6addr** is used to specify a next hop for the routes.

remotev6addr can also be specified instead of **localv6addr**. If a local interface is configured on the same network as **remotev6addr**, this interface address will be used for **localv6addr** if it is unspecified.

Default

none

Context

```
mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement
```

Examples

```
mpbgp on {
    group type internal peeras 64512 {v4u v6u} remotev6addr
        fec0:0:0:ff::d {
            peer 192.0.2.1;
        };
    group type external peeras 64513 {v4u v6u} remotev6addr
        fec0:0:0:ff::e {
            peer 192.0.2.2;
```

```
    };  
};
```

See Also

`localv4addr` on page 472

`localv6addr` on page 474

`remotev4addr` on page 500

routepeer

Name

`routepeer` - specifies the actual time to live (TTL) used on a socket in all cases

Syntax

```
routepeer ;
```

Parameters

none

Description

`routepeer` specifies the actual TTL used on a socket in all cases. In particular, if GateD realizes that two BGP speakers are peering over a single network, GateD automatically sets the `dontroute` option on their peering session. This, in turn, causes the TTL of the packets to be set to 1. `routepeer` prevents the `dontroute` option from being set. If you specify `routepeer`, but don't specify a TTL, and you are directly connected, GateD will set the TTL of your peering session to 1. If you want a TTL greater than 1 for directly connected peers, you must specify both `routepeer` and the `ttl` that you require.

Default

disabled

Context

```
mpbgp group type external statement  
mpbgp group type internal statement  
mpbgp group type routing statement  
mpbgp peer statement
```

Examples

```
[autonomoussystem and routerid omitted]  
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 routepeer ttl 5 ;  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

`ttl` on page 510

sendbuffer

Name

sendbuffer - controls the amount of send buffering asked of the kernel

Syntax

```
sendbuffer buffer_size ;
```

Parameters

buffer_size - an integer from 1 to 65535 inclusive

Description

sendbuffer controls the amount of send buffering asked of the kernel. The maximum supported is 65535 bytes, although many kernels have a lower limit. By default, GateD configures the maximum supported. **sendbuffer** is not needed on normally functioning systems.

Default

```
sendbuffer 65535 ;
```

Context

mpbgp group type external statement

mpbgp group type internal statement

mpbgp group type routing statement

mpbgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
mpbgp on {
    group type external sendbuffer 32768 {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

recvbuffer on page 497

setpref

Name

setpref - allows BGP's **Local_Pref** attribute to be used to set the GateD preference on reception, and allows GateD preference to set the **Local_Pref** on transmission

Syntax

```
setpref metric ;
```

Parameters

metric

Description

setpref allows BGP's **Local_Pref** attribute to be used to set the GateD preference on reception, and allows GateD preference to set the **Local_Pref** on transmission. The **set-pref** *metric* works as a lower limit, below which the imported **Local_Pref** may not set the GateD preference. (For full details, see "Preferences and Route Selection" on page 11 in *Configuring GateD*.)

Default

none

Context

mpbgp group type internal statement

mpbgp group type routing statement

Examples

See "Setpref/Local_Pref Overview" on page 75 in *Configuring GateD*.

See Also

"Preferences and Route Selection" on page 11 in *Configuring GateD*

showwarnings

Name

showwarnings - causes GateD to issue warning messages when receiving questionable BGP updates such as duplicate routes and/or deletions of non-existing routes

Syntax

showwarnings

Parameters

none

Description

showwarnings causes GateD to issue warning messages when receiving questionable MPBGP updates such as duplicate routes and/or deletions of non-existing routes. Normally, these events are silently ignored.

Default

disabled

Context

mpbgp group type external statement
mpbgp group type internal statement
mpbgp group type routing statement
mpbgp peer statement

Examples

```
[autonomoussystem and routerid omitted]
mpbgp on {
    group type external peeras 64512 showwarnings {
        peer 192.0.2.1 ;
    } ;
} ;
[import and export statements omitted]
```

See Also

traceoptions on page 508

traceoptions

Name

traceoptions - specifies the tracing options for this group

Syntax

```
traceoptions trace_options ;
```

Parameters

Trace options include:

- packets** - Trace all BGP packets.
- open** - Trace BGP Open packets.
- update** - Trace BGP Update packets.
- keepalive** - Trace BGP Keepalive packets.
- all** - Trace changes to the GateD routing table.

Description

traceoptions specifies the tracing options for this group. By default, these are inherited from the MPBGP or global trace options. These values may be overridden on the peer statements.

Default

inherited from the global **traceoptions**

Context

mpbgp statement
mpbgp group type routing statement
mpbgp peer statement

Examples

```
[autonomoussystem and routerid omitted]  
mpbgp on {  
    traceoptions packets ;  
    group type external peeras 64512 {  
        peer 192.0.2.1 ;  
    } ;  
} ;  
[import and export statements omitted]
```

See Also

`showwarnings` on page 507

ttl

Name

`tttl` - specifies time to live

Syntax

`tttl tttl ;`

Parameters

`tttl` - `tttl` has two units: seconds and number of hops. Either can be used.

Description

By default, GateD sets the IP TTL for local peers to 1, and the TTL for non-local peers to the default kernel value. The `tttl` option is provided mainly when attempting to communicate with improperly functioning routers that ignore packets sent with a `tttl` of 1. Not all kernels allow the TTL to be specified for TCP connections.

Default

calculated

Context

`mpbgp group type routing` statement

`mpbgp peer` statement

Examples

[autonomoussystem and routerid omitted]

```
mpbgp on {  
    group type external peeras 64512 {  
        peer 192.0.2.1 ; # ttl of 1 (directly connected network)  
        peer 192.168.1.1 routetopeer ttl 2 ;  
    } ;  
} ;
```

[import and export statements omitted]

See Also

`routetopeer` on page 504

Chapter 19

Multicast Source Discovery Protocol (MSDP)

connect-retry-period

Name

connect-retry-period - specifies the number of seconds to wait after a failed connection attempt before trying again to establish a connection to an MSDP peer

Syntax

```
connect-retry-period sec ;
```

Parameters

sec - an integer between 1 and 65535, inclusive

Description

MSDP peering sessions are explicitly configured and the side of the session with the lower IP address initiates the session. The **connect-retry-period** statement specifies the number of seconds to wait after a failed connection attempt, before attempting another connection.

Defaults

```
connect-retry-period 30;
```

Context

msdp statement

Examples

The following configuration sets the **connect-retry-period** to 15 seconds.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
    connect-retry-period 15;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

`msdp` on page 515

`peer` on page 517

default-rpf-peer

Name

default-rpf-peer - specifies the MSDP peer from which Source-Active messages will be accepted, regardless of the RP address they contain

Syntax

```
default-rpf-peer peer_ip;
```

Parameters

peer_ip - the IPv4 address of the peer from which Source-Active messages will be accepted, regardless of the RP address they contain

Description

This statement allows one to specify that Source-Active messages received from a certain peer should be accepted regardless of the RP address contained in the message and regardless of other topology information.

Defaults

The default is to not configure a default peer of any kind.

Context

msdp statement

Examples

In the following configuration, a **default-rpf-peer** is configured such that Source-Active messages from 192.0.2.3 will be accepted regardless of other topology information.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
    peer 192.0.2.1 192.0.2.3;  
    default-rpf-peer 192.0.2.3;  
};
```

See Also

"Multicast Source Discovery Protocol (MSDP)" on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

static-rpf-peer on page 525

keepalive-period

Name

keepalive-period - specifies the frequency with which MSPD KeepAlive messages are transmitted to each peer

Syntax

```
keepalive-period sec ;
```

Parameters

sec - an integer between 75 and 65535, inclusive

Description

MSDP KeepAlive messages must be periodically sent to all MSDP peers. The **keepalive-period** specifies the number of seconds between subsequent KeepAlive messages.

Defaults

```
keepalive-period 75;
```

Context

msdp statement

Examples

The following configuration configures the **keepalive-period** as 90 seconds.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
    keepalive-period 90;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

peer-holdtime on page 519

msdp

Name

msdp - enables or disables the MSDP protocol

Syntax

```
msdp ( on | off )
```

Parameters

on, off - "yes" and "no" are interpreted as "on" and "off" respectively.

Description

The **msdp** statement enables or disables the MSDP protocol. If the **msdp** statement is not specified, MSDP will not run.

Defaults

```
msdp off
```

Context

global

Examples

Example 1

The following configuration specifies an MSDP peering session with 192.0.2.1 and 192.0.2.2 as the local and the remote end, respectively.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
};
```

Example 2

In the following configuration, the MSDP component is disabled.

```
msdp off {  
    peer 192.0.2.1 192.0.2.2;  
};
```

See Also

"Multicast Source Discovery Protocol (MSDP)" on page 115 in *Configuring GateD*

peer on page 517

msdp-draft-6-compatible

Name

msdp-draft-6-compatible - specifies that this router should speak MSDP as specified in draft-ietf-msdp-spec-06

Syntax

```
msdp-draft-6-compatible;
```

Parameters

none

Description

This statement specifies that the router should speak MSDP as specified in draft-ietf-msdp-spec-06.

Defaults

The default is to speak MSDP as specified in draft-ietf-msdp-spec-12.

Context

msdp statement

Examples

The following configuration states that this router should speak MSDP as specified in draft-ietf-msdp-spec-06.

```
msdp yes{
    msdp-draft-6-compatible;
    peer 192.0.2.1 192.0.2.2;
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

peer

Name

peer - specifies the local and remote ends of an MSDP peering session

Syntax

```
peer local_host remote_host mesh id peeras asnum ;
```

Parameters

local_host, *remote_host* - the IPv4 addresses of the local and remote ends of the desired MSDP peering session

mesh *id* - This subclause specifies that the indicated peer is a member of a mesh group number ID. The parameter, *id*, is an integer between 1 and 65535, inclusive.

peeras *asnum* - This subclause indicates the AS number of the remote end. The parameter, *asnum*, is an integer between 1 and 65535, inclusive.

Description

Each desired MSDP peering session is explicitly configured using a **peer** statement. Peers can be configured into groups called "mesh groups", the purpose of which is to reduce flooding of SA-Advertisement messages. If the router receives an SA-Advertisement message from a peer in mesh group n, then it must not forward the message to other peers that are members of mesh group n. If the router receives an SA-Advertisement from a peer that is not in any mesh group at all, then the SA-Advertisement is forwarded to all other peers.

Defaults

The default is for there to be no peering sessions configured. If a **peer** statement is specified without the **mesh** subclause, then peer is not configured as a member of a mesh group. If a **peer** statement is specified without the **peeras** subclause, then the AS number of the remote end cannot be known.

Context

msdp statement

Examples

In the following configuration, 192.0.2.1 and 192.0.2.2 are the peers of an MSDP peering session belonging to mesh group 1. The AS number for the remote end of the peer is 1.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2 mesh 1 peeras 1 ;  
};
```

See Also

"Multicast Source Discovery Protocol (MSDP)" on page 115 in *Configuring GateD*

msdp on page 515

peer-holdtime

Name

peer-holdtime - specifies the number of seconds that a peer session should be maintained in the absence of a KeepAlive message from the peer

Syntax

```
peer-holdtime sec ;
```

Parameters

sec - an integer between 3 and 65535, inclusive

Description

MSDP peers periodically transmit KeepAlive messages as an indication that the peer is still functioning correctly. In the absence of other control messages, if a KeepAlive message is not received within a certain period of time, then the session with the peer will be torn down. This statement specifies the number of seconds that a peer session will be maintained in the absence of any MSDP control messages.

Defaults

```
peer-holdtime 90 ;
```

Context

msdp statement

Examples

The following configuration configures the **peer-holdtime** as 15 seconds.

```
msdp yes{  
    peer 192.0.2.1 192.0.2.2;  
    peer-holdtime 15;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

keepalive-period on page 514

pim-filter

Name

pim-filter - specifies an (S,G) pair which, if learned from the PIM-SM routing fabric, must not be announced in Source-Active or Source-Active-Request-Reply messages.

Syntax

```
pim-filter source_ip masklen length group_ip masklen length;
```

Parameters

source_ip, *length* - a source address prefix which, when combined with the group address prefix, specifies a range of (S,G) pairs that must be filtered.

group_ip, *length* - a group address prefix which, when combined with the source address prefix, specifies a range of (S,G) pairs that must be filtered.

Description

The **pim-filter** statement allows one to specify a set of (S,G) pairs which, if learned from the PIM-SM routing fabric, must not be announced in Source-Active messages. Note that (S,G) pairs covered by the filter will still be added to the MSDP SA cache, but will not appear in Source-Active or Source-Active-Request-Reply messages.

Defaults

The default is to not filter any (S,G) pairs learned from PIM-SM.

Context

msdp statement

Examples

In the following configuration, a **pim-filter** is configured such that any (S,G) pairs learned from PIM-SM fabric that match (*, 227.1.1.1) will be placed in the SA Cache, but will not be announced in Source-Active messages or Source-Active-Request-Reply messages.

```
msdp yes{  
    peer 192.0.2.1 192.0.2.2;  
    pim-filter 0.0.0.0 masklen 0 227.1.1.1 masklen 32;  
};
```

See Also

"Multicast Source Discovery Protocol (MSDP)" on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

sa-filter on page 522

sa-cache-timeout

Name

sa-cache-timeout - specifies the number of seconds that an entry in the SA Cache will be maintained in the absence of a refreshing SA-Advertisement message

Syntax

```
sa-cache-timeout sec;
```

Parameters

sec - an integer between 90 and 65535, inclusive

Description

An entry is created in the MSDP SA Cache when an (S,G) pair is learned of via an MSDP SA-Advertisement message. The **sa-cache-timeout** statement specifies the number of seconds that the entry will remain in the cache unless another SA-Advertisement for the pair is received.

Defaults

```
sa-cache-timeout 210;
```

Context

msdp statement

Examples

In the following configuration, the **sa-cache-timeout** is configured as 250 seconds.

```
msdp yes{  
    peer 192.0.2.1 192.0.2.2;  
    sa-cache-timeout 250;  
};
```

See Also

"Multicast Source Discovery Protocol (MSDP)" on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

sa-filter

Name

sa-filter - specifies an (S,G) pair which, if received in a Source-Active message, must not be forwarded to any MSDP peers.

Syntax

```
sa-filter [ import | export ] source_ip masklen length group_ip masklen
length ;
```

Parameters

source_ip, *length* - A source address prefix which, when combined with the group address prefix, specifies a range of (S,G) pairs that must be filtered. The *source_ip* is an IPv4 address in dotted-quad notation.

group_ip, *length* - A group address prefix which, when combined with the source address prefix, specifies a range of (S,G) pairs that must be filtered. The *group_ip* is an IPv4 class D address in dotted-quad notation.

import - an indication that matching (S,G) pairs must not be communicated to associated PIM-SM components

export - an indication that matching (S,G) pairs must not be forwarded to MSDP peers, or appear in Source-Active-Request-Reply messages

Description

The **sa-filter** statement allows one to specify filters for (S,G) pairs received in Source-Active TLVs received from peers. If the **import** keyword is present, then associated PIM-SM components will not be informed of (S,G) pairs matching the filter. If the **export** keyword is present and the **import** keyword is absent, then matching (S,G) pairs will be used by the configured PIM-SM components, but will not be forwarded to MSDP peers in Source-Active messages or in Source-Active-Request-Reply messages.

Defaults

The default is to not filter any (S,G) pairs.

Context

msdp statement

Examples

Example 1

The following example configures an SA filter such that the associated PIM-SM components will not be informed of any (S,G) pairs received in a Source-Active TLVs that match (*, 227.1.1.1).

```
msdp yes {
```

```
peer 192.0.2.1 192.0.2.2;  
peer 192.0.2.1 192.0.2.3;  
sa-filter import 0.0.0.0 masklen 0 227.1.1.1 masklen 32;  
};
```

Example 2

In the following example, an SA filter is configured such that any (S,G) pairs received in Source-Active TLVs whose source is 192.0.2.4 and group is 227.1.1.1 will be used by the associated PIM-SM component, but will not be forwarded to peers.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
    sa-filter export 192.0.2.4 masklen 32 227.1.1.1 masklen 32;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

pim-filter on page 520

sa-holddown

Name

sa-holddown - specifies the number of seconds that must elapse between subsequent SA-Advertisement messages for an (S,G) pair, in order for the SA-Advertisement messages to be forwarded

Syntax

```
sa-holddown sec;
```

Parameters

sec - an integer between 1 and 65535, inclusive

Description

An entry is created in the MSDP SA Cache when an (S,G) pair is discovered via an MSDP SA-Advertisement message. The message is then forwarded to downstream MSDP peers. In order to mitigate the effects of any SA-Advertisement loops that might develop in the MSDP routing fabric, SA-Advertisements must be rate-limited per (S,G) pair. The **sa-holddown** statement specifies the minimum time that must elapse between forwarding of SA-Advertisement messages containing a given (S,G) pair.

Defaults

```
sa-holddown 30;
```

Context

msdp statement

Examples

The following configuration configures the **sa-holddown** to be 60 seconds.

```
msdp yes{  
    peer 192.0.2.1 192.0.2.2;  
    sa-holddown 60;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

peer on page 517

static-rpf-peer

Name

static-rpf-peer - specifies the MSDP peer from which a Source-Active message for a certain RP will be accepted if none of the other Peer-RPF forwarding rules apply

Syntax

```
static-rpf-peer peer_ip rp_addr_ip;
```

Parameters

peer_ip - the IPv4 address of the peer from which Source-Active messages for RP, *rp_addr_ip*, will be accepted

rp_addr_ip - the IPv4 address of the RP which, if contained in a Source-Active message received from *peer_ip*, will be accepted

Description

This statement allows one to specify that Source-Active messages containing a certain RP address should be accepted from a certain MSDP peer, regardless of other topology information.

Defaults

The default is to not configure a default peer of any kind.

Context

msdp statement

Examples

In the following configuration, a **static-rpf-peer** statement specifies that Source-Active messages containing RP 192.0.2.4 from peer 192.0.2.3 will be accepted.

```
msdp yes {  
    peer 192.0.2.1 192.0.2.2;  
    peer 192.0.2.1 192.0.2.3;  
    static-rpf-peer 192.0.2.3 192.0.2.4;  
};
```

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

default-rpf-peer on page 513

msdp on page 515

peer on page 517

traceoptions

Name

traceoptions - specifies the tracing options for MSDP

Syntax

```
traceoptions trace_options ;
```

Parameters

detail - must be specified before **send** or **receive**. 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.

send or **receive** - limit the tracing to packets sent or received, respectively. If neither is specified, both sent and received packets will be traced.

keepalive - Trace all MSDP KeepAlive messages.

sa-request - Trace all MSDP SA Request messages.

sa-reply - Trace all MSDP SA Reply messages.

sa - Trace all MSDP SA messages.

Description

traceoptions specifies the tracing options for MSDP. By default, these are inherited from the global trace options.

Default

MSDP not traced by default

Context

msdp statement

Examples

See Also

“Multicast Source Discovery Protocol (MSDP)” on page 115 in *Configuring GateD*

msdp on page 515

Chapter 20

Internet Group Management Protocol (IGMP)

enable | disable

Name

enable | **disable** - enables or disables IGMP on this interface or list of interfaces

Syntax

```
enable ; | disable ;
```

Parameters

none

Description

The **enable** statement causes IGMP to be spoken over the interfaces named in the **interface** statement in which the **enable** statement is contained. The **disable** statement specifies that IGMP messages received on any of the interfaces named in the **interface** statement must be ignored.

Defaults

```
enable ;
```

Context

igmp interface statement

Examples

Example 1

The following configuration disables IGMP on all fxp interfaces except for fxp2.

```
igmp on {  
    interface fxp { disable; } ;  
    interface fxp2 { enable; } ;  
};
```

Example 2

The default behavior is that if one or more **interface** statements appear within an **igmp on** or **igmp yes** clause, then IGMP runs on the interfaces named in the **interface** statements. Therefore, the following syntax provides the same configuration as in example 1.

```
igmp on {  
    interface fxp2 { enable; } ;  
};
```

Example 3

The absence of the enable or disable statement implies that the named interfaces should be enabled; thus, the configuration in example 2 can be reduced even further as follows:

```
igmp on {  
    interface fxp2;  
};
```

Example 4

The following example will disable IGMP on all interfaces, not only those mentioned in the **interface** statements below.

```
igmp off {  
    interface fxp2 fxp3;  
};
```

See Also

"Chapter 24 Internet Group Management Protocol (IGMP)" on page 119 in *Configuring GateD*

igmp on on page 529

igmp

Name

igmp - enables or disables the IGMP protocol

Syntax

```
igmp (on | off) [{igmp_statements}];
```

Parameters

on - turns on the IGMP protocol

off - turns off the IGMP protocol

igmp_statements - any of the various commands and options that are available (in this section)

Description

The **igmp** statement specifies whether or not the IGMP protocol is to run on any interfaces. If **igmp yes** or **igmp on** is specified, then IGMP will run on all of the interfaces named in the **interface** statements, contained within the **igmp** statement, subject to the other options contained within the **interface** statements. If no **interface** statements are present, then IGMP will run on all broadcast and multicast-capable or point-to-point interfaces. If **igmp no** or **igmp off** is specified, then IGMP will not run on any interfaces, regardless of the presence of **interface** statements contained within the **igmp** statement.

This statement enables or disables the IGMP protocol. If enabled, IGMP will default to enabling all interfaces that are both broadcast and multicast-capable or point-to-point. These interfaces are identified by the **IFF_BROADCAST** and **IFF_MULTICAST** interface flags. IGMP must be enabled before one of the IP Multicast routing protocols is enabled.

Defaults

```
igmp off
```

Context

global

Examples

Example 1

The following configuration enables IGMP on all interfaces.

```
igmp on ;
```

Example 2

The following configuration disables IGMP on all interfaces.

```
igmp off;
```

Example 3

The following configuration enables IGMP on interface fxp2 only.

```
igmp on {  
    interface fxp2 { enable; };  
};
```

See Also

"Chapter 24 Internet Group Management Protocol (IGMP)" on page 119 in *Configuring GateD*

enable | **disable** on page 527

interface on page 531

interface

Name

interface - selects the interfaces on which the IGMP protocol will be run

Syntax

```
interface interface_list
```

Parameters

interface_list - one or more interface names, including wildcard names (names without a number) and names that can specify more than one interface or address, or the token **all** for all interfaces

Description

The **interface** statement specifies the interfaces on which IGMP will run.

Defaults

If no **interface** statements appear within the **igmp** statement, then the default behavior is to run IGMP on all broadcast and multicast-capable or point-to-point interfaces. If at least one **interface** statement is present, then IGMP will run only on the named interfaces, subject to the options contained within the **interface** statements.

Context

igmp statement

Examples

Example 1

The following configuration specifies that IGMP should run on interface fxp2 only.

```
igmp on {  
    interface fxp2 {enable;} ;  
};
```

Example 2

The following configuration specifies that IGMP should run on all interfaces whose name begins with "fxp".

```
igmp on {  
    interface fxp {enable;} ;  
}
```

See Also

“Chapter 24 Internet Group Management Protocol (IGMP)” on page 119 in *Configuring GateD*

igmp on page 529

last-mem-query-intvl

Name

last-mem-query-intvl - taken together with the value specified in the **robustness** and **max-response-time** statements, **last-mem-query-intvl** specifies the number of seconds to wait for an IGMP Group Membership Report before concluding that no members are present for the indicated group.

Syntax

```
last-mem-query-intvl sec ;
```

Parameters

sec - an integer between the **max-response-time** + 1 and 25, inclusive.

Description

When an IGMP router receives an IGMP Group Leave message on interface *i*, it sends several IGMP Group-Specific Queries. If no Membership Report message is received before the Max Response time of the last Group-Specific Query elapses, then the router concludes that the indicated group has no members reachable via *i*. The **last-mem-query-intvl** statement specifies the amount of time between transmission of subsequent Group-Specific Query messages. The Max Response Time field of each message is set to the value specified in the **max-response-time** statement. The value specified by the **last-mem-query-intvl** statement corresponds to "Last Member Query Interval" as defined in the IGMPv2 protocol specification. The **robustness** statement determines the total number of Group-Specific Queries that are sent.

The **last-mem-query-intvl** statement can appear inside the **igmp** statement but outside of all **interface** statements, as well as within an **interface** statement. If it appears outside of all **interface** statements, then it specifies the value for all interfaces. If the **last-mem-query-intvl** statement appears inside of an **interface** statement, then it specifies the value for all interfaces named in the **interface** statement, overriding any **last-mem-query-intvl** statement that may appear elsewhere.

Defaults

```
last-mem-query-intvl 1 ;
```

Context

igmp statement

igmp interface statement

Examples

Example 1

The following configuration sets the Last Memory Query Interval to five seconds for all interfaces on which IGMP is running. The default value of the **robustness** statement is two and the default value of the **max-response-time** statement is 10 seconds. Therefore,

after receiving a Group Leave message for Group G on fxp1, GateD will send at most two Group-Specific Query messages, one every 5 seconds, each with the Max Response Time field set to 10 seconds. If no Group Membership Report for G is received within 15 seconds (5+10=15), then the router will conclude that there are no members of G reachable via fxp1.

```
igmp on {  
    interface fxp;  
    last-mem-query-intvl 5;  
};
```

Example 2

This will configure fxp2 with a Last Member Query Interval of 3 seconds; fxp3 will have a Last Member Query Interval of 4 seconds; all the rest of the fxp interfaces will have a Last Member Query Interval of 5 seconds.

```
igmp on {  
    interface fxp {  
        enable;  
    };  
    interface fxp2 {  
        enable;  
        last-mem-query-intvl 3;  
    };  
    interface fxp1 {  
        enable;  
        last-mem-query-intvl 4;  
    }  
    last-mem-query-intvl 5;  
};
```

Example 3

This will configure the default Last Member Query Interval to 5 seconds for all interfaces on which IGMP is running. On fxp1, this value is overridden to 4 seconds, and on fxp2, it is overridden to 3 seconds.

```
igmp on {  
    last-mem-query-intvl 5;  
    interface fxp {  
        enable;  
    };  
    interface fxp1 {  
        enable;  
        last-mem-query-intvl 4;  
    }  
};
```

```
};  
interface fxp2 {  
    enable;  
    last-mem-query-intvl 3;  
};  
}
```

See Also

“Chapter 24 Internet Group Management Protocol (IGMP)” on page 119 in *Configuring GateD*

max-response-time on page 536

robustness on page 541

max-response-time

Name

max-response-time - Taken together with the value specified in the **last-mem-query-intvl** and robustness statements, **max-response-time** specifies the number of seconds to wait for an IGMP Group Membership Report before concluding that no members are present for the indicated group.

Syntax

```
max-response-time sec ;
```

Parameters

sec - The amount of time, in seconds, that a router will wait to hear a membership report before removing the group dependency. This is an integer between 1 and MIN(25, Query Interval) inclusive.

Description

The **max-response-time** statement specifies in seconds the value to include in the Max Response Time field of IGMP Group-Specific Query messages. When an IGMP router receives an IGMP Group Leave message on interface *i*, it sends several IGMP Group-Specific Queries. If no Membership Report message is received before the Max Response Time of the last Group-Specific Query elapses, then the router concludes that the indicated group has no members reachable via *i*. The **last-mem-query-intvl** statement specifies the time in seconds between subsequent transmissions of Group-Specific Query messages. The **robustness** statement determines the total number of Group-Specific Queries that are sent.

The argument to **max-response-time** can be any integer between one and either 25 or the Query Interval, whichever is lower. The Query Interval value is set with the **query-interval** statement.

The **max-response-time** statement can appear inside the **igmp** statement, but outside of all **interface** statements, as well as within an **interface** statement. If it appears outside of all **interface** statements, then it specifies the value for all interfaces. If the **max-response-time** statement appears inside of an **interface** statement, then it specifies the value for all interfaces named in the **interface** statement, overriding any **max-response-time** statement that may appear elsewhere.

Defaults

```
max-response-time 10 ;
```

Context

igmp statement

igmp interface statement

Examples

Example 1

The following configuration sets Max Response Time to 220 seconds for all interfaces. The default value of the `last-mem-query-intvl` is one second. Therefore, after receiving a Group Leave message for group G on fxp1, GateD will send at most two Group-Specific Query messages, one every second, each with a Max Response Time set to 220 seconds. If no Group Membership Report for G is received within $1+220=221$ seconds, then the router will conclude that there are no members of group G reachable via fxp1.

Set the max-response time for all interfaces to 220 seconds.

```
igmp on {
    interface all {
        enable;
    };
    max-response-time 220;
}
```

Example 2

Set the max-response time to 220 seconds for all interfaces on which IGMP is running. On fxp1, this value is overridden to 30 seconds.

```
igmp on {
    interface all {
        enable;
    };
    max-response-time 220;
    interface fxp1 {
        max-response-time 30;
    };
}
```

See Also

"Chapter 24 Internet Group Management Protocol (IGMP)" on page 119 in *Configuring GateD*

`last-mem-query-intvl` on page 533

`robustness` on page 541

nosend

Name

nosend - allows the interface to receive, but prevents it from sending, any IGMP packets

Syntax

nosend ;

Parameters

none

Description

nosend allows the interface to receive, but prevents it from sending, any IGMP packets. **nosend** permits the construction of a kernel Virtual InterFace (VIF) for the configured interface and, thus, allows GateD to subdue kernel upcalls (for example, IGMP NOCACHE upcalls) in some (older) kernels.

Defaults

disabled

Context

igmp interface statement

Examples

This example configures IGMP to run all fxp interfaces, but prevents the sending of IGMP messages.

```
igmp on {  
    interface fxp {  
        enable;  
    };  
    interface fxp2 {  
        enable;  
        nosend;  
    };  
}
```

See Also

"Chapter 24 Internet Group Management Protocol (IGMP)" on page 119 in *Configuring GateD*

igmp on page 529

query-interval

Name

query-interval - specifies the time between the transmission of General Query messages in seconds

Syntax

```
query-interval sec ;
```

Parameters

sec - an integer between 1 and 65535 inclusive

Description

The Designated Querier on a subnet periodically sends General Query messages to verify the state of group memberships. The frequency with which these messages are sent is specified with the **query-interval** statement.

The **query-interval** statement can appear inside the **igmp** statement, but outside of all **interface** statements, as well as within an **interface** statement. If it appears outside of all **interface** statements, then it specifies the value for all interfaces. If the **query-interval** statement appears inside of an **interface** statement, then it specifies the value for all interfaces named in the **interface** statement, overriding any **query-interval** statement that may appear elsewhere.

Defaults

```
query-interval 125 ;
```

Context

igmp statement

igmp interface statement

Examples

Example 1

The following configuration sets the query-interval for interface fxp2 to 55 seconds. The query-interval for all remaining interfaces defaults to 125 seconds.

```
igmp yes {  
    interface fxp {  
        enable;  
    };  
    interface fxp2 {  
        enable;  
        query-interval 55;  
    }  
}
```

```
    };  
}
```

Example 2

The following configuration sets the query-interval for fxp2 to 120 seconds and the query-interval for all remaining interfaces to 130 seconds.

```
igmp yes {  
    interface fxp {  
        enable;  
    };  
    interface fxp2 {  
        enable;  
        query-interval 120;  
    };  
    query-interval 130;  
}
```

See Also

“Chapter 24 Internet Group Management Protocol (IGMP)” on page 119 in *Configuring GateD*

`igmp` on page 529

robustness

Name

robustness - allows for tuning of the IGMP protocol to accommodate a lossy subnet

Syntax

```
robustness value ;
```

Parameters

num - an integer between 2 and 65535 inclusive

Description

The **robustness** statement determines the number of IGMP General Query messages sent when a router first comes up, as well as the number of Group-Specific Query messages sent in response to receiving a Group Leave message. On a given network, IGMP is "robust" to the loss of "robustness - 1" messages. Per the IGMP protocol specification, the robustness value should be greater than 1.

The **robustness** statement can appear inside the **igmp** statement, but outside of all **interface** statements, as well as within an **interface** statement. If it appears outside of all **interface** statements, then it specifies the value for all interfaces. If the **robustness** statement appears inside of an **interface** statement, then it specifies the value for all interfaces named in the **interface** statement, overriding any **robustness** statement that may appear elsewhere.

Defaults

```
robustness 2 ;
```

Context

igmp statement

igmp interface statement

Examples

Example 1

The following configuration sets the robustness for interface fxp2 to three. All other interfaces have the default robustness of two.

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

```
        robustness 3;
    };
}
```

Example 2

The following configuration sets all the default robustness for all interfaces to three. The robustness for interface fxp2 is overridden to be four.

```
igmp yes {
    interface fxp {
        enable;
    };
    interface fxp2 {
        enable;
        robustness 4;
    };
    robustness 3;
}
```

See Also

“Chapter 24 Internet Group Management Protocol (IGMP)” on page 119 in *Configuring GateD*

`last-mem-query-intvl` on page 533

`max-response-time` on page 536

traceoptions

Name

traceoptions - specifies the tracing options for this group

Syntax

```
traceoptions trace_options ;
```

Parameters

Trace options include:

packets - Trace all IGMP packets.

query - Trace IGMP Membership Query messages.

report - Trace IGMP Membership Report messages.

leave - Trace IGMP Leave messages.

mtrace - Trace Mtrace messages. Mtrace is defined in draft-ietf-idmp-traceroute-ipm-05.txt.

all - Trace all IGMP messages.

Description

traceoptions specifies the tracing options for IGMP. By default, these are inherited from the global trace options.

Default

inherited from global traceoptions

Context

igmp statement

Examples

Example 1

In the following example, all IGMP events will be logged to the file, /var/tmp/igmp.log. This includes timer events and exceptional events, as well as sending and receiving IGMP messages.

```
igmp yes {  
    traceoptions "/var/tmp/igmp.log" all;  
    interface fxp {  
        enable;  
    };  
    interface fxp2 {  
        enable;  
    };  
}
```

```
        robustness 3;
    };
};
```

Example 2

In the following example, only IGMP Membership Report and Membership Query messages that are sent and/or received are logged to the file, /var/tmp/igmp.log.

```
igmp yes {
    traceoptions "/var/tmp/igmp.log" report query;
    interface fxp {
        enable;
    };
    interface fxp2 {
        enable;
        robustness 3;
    };
};
```

Example 3

In the following example, only IGMP Membership Reports that are sent, and Query messages that are received, are logged to the file, /var/tmp/igmp.log.

```
igmp yes {
    traceoptions "/var/tmp/igmp.log" send report recv query;
    interface fxp {
        enable;
    };
    interface fxp2 {
        enable;
        robustness 3;
    };
};
```

See Also

“Chapter 24 Internet Group Management Protocol (IGMP)” on page 119 in *Configuring GateD*

“Mtrace” as defined in draft-ietf-idmp-traceroute-ipm-05.txt

`traceoptions` on page 3

Chapter 21

Multicast Statement

boundary

Name

boundary - configures administratively scoped group boundaries on the indicated interface(s)

Syntax

```
boundary group-address [ ( mask mask ) | ( masklen number ) ]  
                interface_list
```

Parameters

group-address - a multicast group address which, when combined with **mask** or **masklen**, specifies the set of multicast groups for which administrative boundaries are to be configured

mask *mask* - a netmask in dotted-quad notation, which, when combined with *group-address*, specifies the set of multicast groups for which administrative boundaries are to be configured.

masklen *number* - an alternative method for specifying the mask associated with *group-address*. In this method, the mask length is specified in bits.

interface_list - 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

Description

boundary is used to configure administratively scoped group boundaries on the indicated interface(s).

Defaults

none

Context

multicast statement

Examples

The example below configures interface `le1` with a time-to-live (TTL) threshold of 16 and sets a boundary for 225.255/16 on it.

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

See Also

`multicast` on page 548

interface

Name

interface - used to specify TTL thresholds

Syntax

```
interface interface_list [ threshold number ]
```

Parameters

interface_list - 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

threshold *number* - number to indicate TTL for packet

Description

On multicast capable interfaces, **interface** is used to specify TTL thresholds and rate limits.

Defaults

```
interface interface_list threshold 1
```

Context

multicast statement

Examples

```
multicast {  
    interface le1 threshold 16  
    join 239.1.2.3 le1;  
};
```

See Also

multicast on page 548

multicast

Name

multicast - overrides default multicast options

Syntax

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

Parameters

interface - used to specify TTL thresholds

boundary - configures administratively scoped group boundaries on the indicated interface(s)

Description

The **multicast** statement is not required for multicast routing to work. It is used only for overriding default options.

Defaults

none

Context

global statement

Examples

```
multicast {  
    interface le1 threshold 16  
    boundary 239.255.0.0 masklen 16 le1;  
};
```

See Also

"Multicast Statement" on page 123 in *Configuring GateD*

Chapter 22

Mstatic Statement

disable

Name

disable - disables the configuration information on the interfaces to which it refers

Syntax

```
disable ;
```

Parameters

none

Description

disable is used to disable a list of one or more interfaces. Its primary use is to disable a specific interface that might otherwise be enabled by a less specific interface reference.

Defaults

```
enable;
```

Context

```
mstatic interface statement
```

Examples

The example below configures group membership for 224.1.1.1 on all multicast-capable interfaces except for the interface named fxp0.

```
mstatic yes {  
    interface all {  
        join 224.1.1.1 ;  
    };  
    interface fxp0 {  
        disable ;  
    };  
};
```

See Also

`enable` on page 551

`mstatic` on page 554

enable

Name

enable - enables the configuration information on the interfaces to which it refers

Syntax

```
enable ;
```

Parameters

none

Description

enable is used to enable the configuration on a list of one or more interfaces. Because this is the default, it is not strictly required.

Defaults

```
enable;
```

Context

```
mstatic interface statement
```

Examples

```
mstatic yes {  
    interface fxp0 {  
        enable ;  
        join 224.1.1.1;  
    };  
};
```

See Also

disable on page 549

mstatic on page 554

interface

Name

interface - specifies a list of one or more interfaces on which group membership information is to be configured

Syntax

```
interface interface_list
```

Parameters

interface_list - one or more interface names, including wildcard names (names without a number) and names that can specify more than one interface or address, or the token **all** for all interfaces

Description

interface is used to specify a list of one or more interfaces on which group membership information is to be configured.

Defaults

The default is to have no interfaces configured.

Context

mstatic statement

Examples

The following example configures group membership information on three interfaces (fxp0, fxp1, and fxp2).

```
mstatic yes {  
    interface fxp0 fxp1 fxp2 {  
        join 224.1.1.1;  
    };  
};
```

See Also

mstatic on page 554

join

Name

`join` - configures group membership information on the indicated interfaces

Syntax

```
join group-address ;
```

Parameters

group-address - a multicast group address that specifies a multicast group for which branches are to be grafted

Description

`join` is used to configure group membership information on the indicated interface(s). Because `join` prevents pruning, static joins should be used only in exceptional circumstances. Note that `join` does not cause GateD to join the indicated group(s) using, for example, IGMP. It only causes GateD to act as if downstream members are present and have joined the group.

Defaults

none

Context

`mstatic interface` statement

Examples

```
mstatic yes {  
    interface le1 {  
        join 239.1.2.3 ;  
    };  
};
```

See Also

`mstatic` on page 554

mstatic

Name

mstatic - configures group membership information on a set of interfaces

Syntax

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

Parameters

interface - specifies a list of one or more interfaces on which group membership information is to be configured

enable | disable - enables or disables the configuration information on the interfaces to which it refers

join - configures group membership information

Description

The **mstatic** statement configures group membership information on a set of interfaces.

Defaults

none

Context

global statement

Examples

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

See Also

"Mstatic Statement" on page 125 in *Configuring GateD*

Chapter 23

Routing Information Protocol, next generation (RIPng)

defaultmetric

Name

defaultmetric - defines the metric Multi-Exit Discriminator (MED) used when advertising routes via RIPng

Syntax

```
defaultmetric metric ;
```

Parameters

metric - a RIPng metric from 0 to 15

Description

When RIPng learns a route from another protocol, the metrics of the two protocols are almost certainly incompatible. Most protocols have a much larger metric space (for example, 0 to 65535) than RIPng does, so there must be a mechanism for specifying the initial metric that the RIPng advertiser will use.

defaultmetric is used to set the default metric for advertising into the RIPng cloud routes learned from other protocols, including **static**. To set this metric for exporting routes to RIPng on a per-protocol basis, use the **export** statement.

Default

```
defaultmetric 1 ;
```

Context

ripng statement

Examples

The following example sets the default metric for routes exported to RIPng to be 2.

```
ripng on {  
    defaultmetric 2 ;  
};
```

See Also

`export` on page 623

`ripng` on page 568

expire-time

Name

expire-time - sets the expiration time for routes received via RIPng

Syntax

```
expire-time expire_time;
```

Parameters

expire_time - an integer from 5 to 180, inclusive

Description

expire-time is initialized when a route is established, and any time an update message is received for the route. If *expire_time* seconds elapse from the last time the **expire-time** was initialized, the route is considered to have expired, and the deletion process begins for that route. This is a **group** option in the **ripng** clause. For example, it is used in the same context as **preference**.

Defaults

```
expire-time 180;
```

Context

ripng statement

Examples

```
ripng yes {  
    expire-time 100;  
    interface fxp0;  
};
```

See Also

update-time on page 575

interface

Name

interface - controls various attributes of sending RIPng on specific interfaces

Syntax

```
interface interface_list [ noripin ] | [ ripin ] [ noripout ] |  
[ ripout ] [ metricin ] | [ metricout ] ;
```

Parameters

interface_list - 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

noripin - specifies that RIPng packets received via the specified interface will be ignored.

ripin - specifies that RIPng packets on all non-loopback interfaces will be listened to. Specifying **ripin** may be necessary when **noripin** is used on a wildcard interface descriptor.

noripout - specifies that no RIPng packets will be sent on the specified interfaces. The sending of RIPng on point-to-point interfaces must be manually configured.

ripout - Specifying **ripout** is necessary to send RIPng on point-to-point interfaces and can be necessary when **noripin** is used on a wildcard interface descriptor.

metricin - sets an additional metric on incoming RIPng routes

metricout - specifies an additional cost to be added to outgoing routes

Description

interface controls various attributes of sending RIPng on specific interfaces. It is used both to set interface-specific parameters in RIPng and to determine on which interfaces RIPng will run. By default, RIPng will run on all interfaces; however, if any specific interfaces (or subsets of interfaces) are explicitly configured with **interface**, then all non-configured interfaces will not run RIPng.

If multiple interfaces (either physical or logical) have addresses on the same subnet, then (regardless of configuration) RIPng will send updates only on the first one for which RIPng is configured to do so.

Although it is possible to specify a loopback interface or loopback address in an interface statement, RIPng will not normally send packets to a loopback. To override this behavior, use **source-gateways** with the loopback address included in the *gatewaylist*.

Notes:

- When specifying a link-local address as the interface, the interface index must be stripped from the interface.
- If there are multiple interfaces configured on the same subnet, RIPng updates will only be sent from the first one for which RIPng output is configured. This limitation is required because of the way the UNIX kernel operates.

Default

```
interface interface_list ripin ripout ;
```

Context

ripng statement

Examples

Example 1

The following will configure RIPng to run on just the eth0 interface (and not any others).

```
ripng on {  
    interface eth0;  
};
```

Example 2

Alternatively, this example will set RIPng to run on all interfaces except eth0.

```
ripng on {  
    interface all ;  
    interface eth0 noripin noripout;  
};
```

See Also

`metricin` on page 562

`metricout` on page 564

`noripin` on page 570

`noripout` on page 571

`ripng` on page 568

“Chapter 5 Interface Statements” on page 15 of the *GateD Command Reference Guide*

“Chapter 7 Interface Statement” on page 23 of *Configuring GateD*

metricin

Name

metricin - sets an additional metric on incoming RIPng routes

Syntax

metricin *ripngmetric*

Parameters

ripngmetric - a number signifying hop count, from 0 to 15

Description

It is often the case that a router should prefer routes received on one set of interfaces over those received on another. For example, given two point-to-point links, one can be more expensive than the other and should, therefore, be less preferred. **metricin** is used for exactly this purpose: to make routes learned from certain interfaces less preferable.

metricin is the default manner by which RIPng increments hop count. That is to say, RIPng works by adding a hop every time a route is received and before it is sent back out to other interfaces. This implementation adds the hop when the route is received (for example, before decisions regarding whether the route should be used are made). By default, a metric of 1 plus the kernel interface metric is added as the hop count. Normally, this interface metric is zero, but some operating systems allow it to be specified on interface configuration. If **metricin** is explicitly given, it is added as an absolute value (for example, without the interface metric).

Defaults

If left unspecified, a metric of 1 plus the kernel interface metric (if any) is the default.

Context

ripng interface statement

Examples

Example 1

To override any specified interface metric, the following adds exactly 1 to the metric of all received routes.

```
ripng on {  
    interface all metricin 1;  
};
```

Example 2

In somewhat more normal usage, the following example increases the cost of routes by 2 that are received over a point-to-point link more than those received on other interfaces.

```
ripng on {
```

```
    interface all metricin 1;  
    interface ppp0 metricin 3;  
};
```

See Also

`interface` on page 560

`metricout` on page 564

`ripin` on page 570

`ripout` on page 571

metricout

Name

metricout - specifies an additional cost to be added to outgoing routes

Syntax

metricout *ripngmetric*

Parameters

ripngmetric - a number signifying hop count, from 0 to 15

Description

Normally, this RIPng implementation adds to the hop count only on incoming routes. There are times, however, when the user wants to cause other routers not to prefer routes from a given origin. For example, if the router is a backup router, it might be desirable for its routes to always be less preferred. **metricout** accomplishes this by adding to the RIPng metric on top of any metric specified by **metricin** before RIPng updates are sent out the specified interface.

Defaults

metricout 0 ;

Context

ripng interface statement

Examples

Example 1

If this router (host 128) is acting as backup on 192.168.10/24 for all other interfaces, the following will add 2 to all metrics advertised out the 192.168.10.128 interface.

```
ripng on {  
    interface all ripin ripout;  
    interface 192.168.10.128 metricout 2;  
};
```

Example 2

The following example has no effect. It is contrived to point out that **metricout** does not impact routing based on the receipt of updates, so if updates do not go out an interface on which **metricout** is specified, the behavior of the entire network will be identical to what it would have been without **metricout** being issued.

```
ripng on {  
    interface all ripin ripout;  
    interface eth0 ripin noripout metricout 15;
```

```
};
```

See Also

`interface` on page 560

`metricin` on page 562

`noripin` on page 570

`noripout` on page 571

`ripin` on page 570

`ripout` on page 571

preference

Name

preference - sets the preference for RIPng routes

Syntax

preference *ripngpreference*

Parameters

ripngpreference - an assigned integer between 0 (directly connected) and 255 (for EGP)

Description

preference specifies how active routes that are learned from RIPng (compared to other protocols) 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. **preference** can be used to change the default value for RIPng. **preference** can be overridden by a **preference** value specified in import policy.

Default

preference 100 ;

Context

ripng statement

Examples

Example 1

The following example makes RIPng routes more preferable than IS-IS routes, but less preferable than OSPF routes.

```
ripng on {  
    preference 13;  
};
```

Example 2

The following example makes RIPng routes less preferable than OSPFase routes, but still more preferable than BGP routes.

```
ripng on {  
    preference 160;  
};
```

See Also

“Preferences and Route Selection” on page 11 in *Configuring GateD*

ripng on page 568

ripng

Name

ripng - an implementation of a distance-vector routing protocol for local networks

Syntax

```
ripng ( on | off ) [ { ripargs } ] ;
```

Parameters

ripngargs - RIPng configuration parameters, which are described throughout this chapter.

Description

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.

Default

ripng is run on all interfaces.

Context

global

Examples

The following example configures the identical behavior as not having a `gated.conf` file would (run RIPng on all interfaces).

```
ripng on ;
```

See Also

`aggregate` on page 673

`dampen-flap` on page 707

`export` on page 623

“Routing Information Protocol, next generation (RIPng)” on page 127 in *Configuring GateD*

ripin, noripin

Name

ripin, **noripin** - specifies whether RIPng will listen to RIPng updates

Syntax

ripin
noripin

Parameters

none

Description

ripin specifies that RIPng will process RIPng updates received on a given interface. Since this is also the default, it is not normally required, except to override a **noripin** specified on a wildcard list of interfaces. **noripin** does exactly the opposite. Although it would almost certainly be a misconfiguration, it is important to note that RIPng can send RIPng updates on a superset of those interfaces on which it receives updates. This can be a valid configuration if, for example, the user receives RIPng updates from an ISP, redistributing those onto the LAN, and does not want to send the LAN topology back to the ISP. But this would be highly unusual.

Defaults

ripin

Context

ripng interface statement

Examples

This somewhat contrived example processes RIPng updates received on all eth devices except eth0.

```
ripng on {  
    interface eth ripin;  
    interface eth0 noripin;  
};
```

See Also

interface on page 560

metricin on page 562

metricout on page 564

ripout on page 571

ripout, noripout

Name

`ripout`, `noripout` - specifies whether RIPng will send RIPng updates

Syntax

`ripout`
`noripout`

Parameters

none

Description

`noripout` specifies that RIPng updates should not be sent on the specified interfaces.
`ripout` specifies the converse and is the default on broadcast-enabled interfaces.

Defaults

`ripout` on broadcast interfaces

`noripout` on nonbroadcast interfaces (including point-to-point interfaces)

Context

`ripng interface` statement

Examples

Example 1

The following example specifies that RIPng updates should not be sent on any eth interfaces except eth0.

```
ripng on {  
    interface eth noripout;  
    interface eth0 ripout;  
};
```

Example 2

The following example specifies that RIPng updates should be sent on all eth and ppp interfaces.

```
ripng on {  
    interface eth ripout;  
    interface ppp ripout;  
};
```

See Also

`interface` on page 560

`metricin` on page 562

`metricout` on page 564

`ripin` on page 570

traceoptions

Name

traceoptions - specifies the tracing options for RIPng

Syntax

traceoptions *trace_options*

Parameters

Trace options include:

packets - Trace all RIPng packets.

request - Trace RIPng information request packets, such as **request**, **poll**, and **pollentry**.

response - Trace RIPng **response** packets, which are the type of packet that actually contains routing information.

other - Trace any other type of packet. The only valid ones are **trace_on** and **trace_off**, both of which are ignored.

Description

traceoptions specifies the tracing options for this group. By default, these are inherited from the global trace options.

Default

inherited from global **traceoptions**

Context

ripng statement

Examples

Example 1

```
traceoptions none;
```

Example 2

```
traceoptions /var/tmp/ripng_peer1 detail packets;
```

Example 3

```
traceoptions receive request;
```

Example 4

```
traceoptions send response;
```

See Also

`ripng` on page 568

update-time

Name

`update-time` - sets the update time for unsolicited route response

Syntax

```
update-time update_time;
```

Parameters

update_time - an integer from 0 to 30, inclusive

Description

This is a group option in the RIPng clause. For example, it is used in the same context as **preference**.

Defaults

```
update-time 30;
```

Context

ripng statement

Examples

```
ripng on {  
    update-time 20;  
    interface fxp0;  
};
```

See Also

`expire-time` on page 559

Chapter 24

Route Filtering

all

Name

all - matches anything

Syntax

```
[ inet6 | inet ] all [ exact | refines | ( between lower and upper ) ]
```

Parameters

none

Description

all matches anything. **all** is equivalent to:

```
0.0.0.0 mask 0.0.0.0
```

and

```
::/0 if in an IPv6 context or if inet6 all is specified.
```

It includes **default**.

Defaults

none

Context

route filter statement

Examples

```
import proto bgp as 201 {  
    all;  
};
```

The above example is equivalent to:

```
import proto bgp as 201 {  
    inet all ;  
}
```

```
        inet6 all ;  
    } ;
```

See Also

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*
network on page 583

between

Name

between - specifies that the mask of the destination must be as or more specific than the lower limit and no more specific than the upper limit

Syntax

between *lower* **and** *upper*

Parameters

lower - the lower bound on matched mask lengths

upper - the upper bound on matched mask lengths

Description

between specifies that the mask of the prefix to match must be as or more specific (for example, as long as or longer) than the lower limit and no more specific (for example, as long as or shorter) than the upper limit. *lower* must be greater than or equal to the filter masklen, and lower must be greater than or equal to upper. Both must be less than or equal to the maximum mask length for the address family. Note that **exact** and **refines** are both special cases of **between**.

between cannot be used with **all** in contexts where **all** refers to both IPv4 and IPv6 addresses. In such cases, **inet all between** and/or **inet6 all between** must be specified.

Defaults

none

Context

route filter statement

Examples

```
import proto bgp as 201 {  
    0.0.0.0 between 24 and 25 restrict;  
    inet all;  
};
```

See Also

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*

network on page 583

default

Name

`default` - matches the `default` route

Syntax

```
[ inet6 | inet ] default
```

Parameters

none

Description

`default` matches the `default` route. To match, the address must be the default address and the mask must be all zeros. `default` is equivalent to:

```
0.0.0.0 mask 0.0.0.0 exact
```

and

```
:: mask :: exact
```

 in an IPv6 context or if `inet6 all` is specified.

Defaults

none

Context

route filter statement

Examples

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

See Also

"Chapter 28 Route Filtering" on page 129 of *Configuring GateD*

`network` on page 583

exact

Name

exact - specifies that the mask of the destination must match the supplied mask exactly

Syntax

exact

Parameters

none

Description

exact specifies that the mask of the destination must match the supplied mask exactly.
exact is used to match a network, but no subnets or hosts of that network.

Defaults

none

Context

route filter statement

Examples

The following example aggregates only the single subnet route to 223.1/25 and all routes more specific than 223.2/24, but not 223.2/24 itself.

```
aggregate 223.0.0.0/8 {  
    proto static {  
        223.1/25 exact;  
        223.2/24 refines;  
    };  
};
```

See Also

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*

network on page 583

host

Name

host - matches the specific host

Syntax

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

Parameters

host - the DNS name or IP address of a host (a maximum length prefix). If a DNS name is specified, GateD attempts to resolve the name to an IP address. **inet6** indicates that a host name is to be resolved as an IPv6 address.

Description

host matches the specific host. To match, the address must exactly match the specified **host** and the network mask must be a host mask (for example, all 1's). **host** is equivalent to:

```
host mask 255.255.255.255
```

or in the IPv6 case:

```
host masklen 128
```

Defaults

none

Context

route filter statement

Examples

The example below imports a RIP route to all addresses except the single address, 10.1.2.3/32.

```
import proto rip {  
    all;  
    host 10.1.2.3 restrict;  
};
```

See Also

"Chapter 28 Route Filtering" on page 129 of *Configuring GateD*

network on page 583

network

Name

network - matches subnets of the specified address and mask

Syntax

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

Parameters

network - the network address to match

mask *mask* - the network mask to use for the match

masklen *number* - the length of the network mask to use for the match

lower - the lower bound on matched mask lengths

upper - the upper bound on matched mask lengths

Description

A route matches the configured filter if the route's address matches that of the filter up to the length of the filter's mask. The route's mask must be at least as long as that of the filter in order to match. Furthermore, **exact** indicates that a matched address mask must be equal to that of the filter. **refines** indicates that a matched route's mask must be longer than the filter's. **between** indicates that a matched route's length must be within the specified *lower* and *upper* bounds.

Defaults

If a network is specified without a **mask** or **masklen**, and the address type is **inet**, the mask defaults to the natural mask for the network.

Context

route filter statement

Examples

The following example aggregates only the single subnet route to 223.1/25 and all routes more specific than 223.2/24, but not 223.2/24 itself.

```
aggregate 223.0.0.0/8 {
    proto static {
        223.1/25 exact;
        223.2/24 refines;
    };
};
```

See Also

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*
martians on page 90

refines

Name

refines - specifies that the mask of the destination must be more specific than the filter mask

Syntax

refines

Parameters

none

Description

refines specifies that the mask of the destination must be more specific (i.e., longer) than the filter mask. **refines** is used to match subnets and/or hosts of a network, but not the network.

Defaults

none

Context

route filter statement

Examples

The following example imports all routes except default (0.0.0.0/0) and 10.1.2.3 via RIP.

```
import proto rip {  
    all refines;  
    host 10.1.2.3 restrict;  
};
```

See Also

all on page 577

"Chapter 28 Route Filtering" on page 129 of *Configuring GateD*

network on page 583

route filter

Name

route filter - matches route filters

Syntax

```
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
```

Parameters

includes all those parameters discussed in this chapter

Description

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 on **martians**, and in **import** and **export** statements.

In most cases, you can specify additional parameters relevant to the context of the filter. For example, on a **martian** statement you can specify the **allow** keyword; on an **import** statement you can specify a **preference**; and on an **export** statement you can specify a **metric**.

Three optional types of matching used to filter routes are **exact**, **refines**, and **between**.

Defaults

If a network is specified without a **mask** or **masklen**, and the address type is **inet**, the mask defaults to the natural mask for the network.

Context

aggregate statement

martians statement

import statement

export statement

Examples

The following example shows how to set up a route filter for BGP-import that allows all IPv4 networks with a masklen less than 19 to pass.

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

See Also

“Chapter 28 Route Filtering” on page 129 of *Configuring GateD*
martians on page 90

Chapter 25

Matching AS Paths

aspath

Name

aspath - specifies that an AS_PATH matching the *aspath_regexp* with the specified origin is matched

Syntax

```
aspath aspath_regexp
```

Parameters

aspath_regexp - a BGP AS path regular expression. See “AS Path Regular Expressions” on page 131 in *Configuring GateD* for more information on regular expressions.

Description

aspath specifies that an AS matching the *aspath_regexp* with the specified origin is matched.

Defaults

none

Context

import statement

subclauses of **export** statement

Examples

Example 1

```
import proto bgp aspath "(4)" origin any {  
    all;  
};
```

Example 2

```
import proto bgp aspath "(1+)" origin any {  
    all;  
};
```

See Also

"AS Path Regular Expressions" on page 131 in *Configuring GateD*
`origin` on page 591

origin

Name

origin - specifies whether the route was learned from an EGP or an IGP source

Syntax

```
origin ( [ any ] | [ igp ] | [ egp ] | [ incomplete ] )
```

Parameters

any
igp
egp
incomplete

Description

An **origin** of **igp** indicates the route was learned from an Intra-Domain Routing Protocol and is most likely complete. An **origin** of **egp** indicates the route was learned from the EGP protocol, and the path is most likely not complete. When the route is learned from another source, an **origin** of **incomplete** is used. An **origin** of **any** can be used to match any origin.

Defaults

none

Context

aspath statement

Examples

Example 1

```
import proto bgp aspath "(4)" origin any {  
    all;  
};
```

Example 2

```
import proto bgp aspath "(1+)" origin any {  
    all;  
};
```

See Also

aspath on page 589

"Chapter 29 Matching AS Paths" on page 131 in *Configuring GateD*

Chapter 26

BGP Communities

comm

Name

`comm` - specifies a set of communities to match

Syntax

```
comm { community_list }
```

Parameters

`{ community_list }` - specifies the set of communities that are to be matched

Description

The `comm` clause is used to specify a set of communities that must be matched for a route to be considered a match.

Defaults

The default is to ignore the communities attribute.

Context

BGP import statements or BGP subclauses of export statements

Examples

The following example exports to BGP peers in AS 65534 all routes that were learned from BGP peers in AS 65533 with the community 65533 101:

```
export proto bgp as 65534 {  
    proto bgp as 65533 comm { comm-split 65533 101; } {  
        all;  
    };  
};
```

See Also

“Chapter 30 BGP Communities” on page 133 in *Configuring GateD*

comm-add on page 595

comm-delete on page 596

community_list on page 597

export statements on page 623

import statements on page 605

comm-add

Name

comm-add - specifies a set of communities to be added on export

Syntax

```
comm-add { community_list }
```

Parameters

{ *community_list* } - specifies the set of communities to be added to routes matching this export clause

Description

The **comm-add** clause is used to specify a set of communities that are to be added to route advertisements matching a particular BGP export statement.

Defaults

The default is to not add communities to outgoing route advertisements.

Context

BGP export statements

Examples

The following example adds an arbitrary community (0x123 0x231) to BGP routes learned from peers in AS 65533 when they are exported to BGP peers in AS 65534:

```
export proto bgp as 65534 comm-add { comm-hex 0x123 0x231; } {  
    proto bgp as 65533 {  
        all;  
    };  
};
```

See Also

"Chapter 30 BGP Communities" on page 133 in *Configuring GateD*

community_list on page 597

comm on page 593

comm-delete on page 596

export statements on page 623

import statements on page 605

comm-delete

Name

comm-delete - specifies a set of communities to be removed on export

Syntax

```
comm-delete { community_list }
```

Parameters

{ *community_list* } - specifies the set of communities to be deleted from routes matching this export clause

Description

The **comm-delete** clause is used to specify a set of communities that are to be removed from routes being advertised as a result of matching the export statement containing the **comm-delete** clause.

Defaults

The default is to not remove communities from out-going route advertisements.

Context

BGP export statements

Examples

The following example removes an arbitrary community (0x123 0x231) from all routes learned from BGP peers in AS 65534 when they are advertised to BGP peers in AS 65533:

```
export proto bgp as 65533 comm-delete { comm-hex 0x123 0x231; } {  
    proto bgp as 65534 {  
        all;  
    };  
};
```

See Also

“Chapter 30 BGP Communities” on page 133 in *Configuring GateD*

export statements on page 623

import statements on page 605

community_list

Name

community_list - a list of communities that is used for matching or modifying AS paths

Syntax

```
community none ;
community no-export ; ...
community no-advertise ; ...
community no-export-subconfed ; ...
comm-hex hex-number hex-number ; ...
comm-split autonomous_system community-id ; ...
```

Parameters

community none - This parameter specifies the empty set of communities. It is useful only in the comm statement where matches are being attempted. When used in that context, if a route has any communities associated with it, then it does not match; otherwise, it does match.

community no-export - Specifies the well-known community NO_EXPORT as defined in RFC 1997. Routes tagged with this community are not to be exported outside of a confederation boundary.

community no-advertise - Specifies the well-known community NO_ADVERTISE as defined in RFC 1997. Routes tagged with this community are not to be advertised to any other peers.

community no-export-subconfed - Specifies the well-known community NO_EXPORT_SUBCONFED as defined in RFC 1997. Routes tagged with this community are not to be advertised to external peers, even if they are within the same confederation.

comm-hex hex-number hex-number - This parameter specifies any arbitrary community that is the concatenation of the two sixteen-bit numbers specified.

comm-split autonomous_system community-id - This parameter specifies a community that is the concatenation of the AS number ASN and the arbitrary sixteen-bit number.

Description

A *community_list* specifies a series of communities that are to be matched against a route, added to a route on export, or deleted from a route on export. Any number of communities can be specified in a single community list, in any order.

Defaults

none

Context

comm-add

`comm-delete`

`comm`

Examples

The following example exports all static routes to BGP peers in AS 65534, adding the well-known NO_EXPORT community:

```
export proto bgp as 65534 comm-add { community no-export; } {  
    proto static {  
        all;  
    };  
};
```

See Also

“Chapter 30 BGP Communities” on page 133 in *Configuring GateD*

`comm-add` on page 595

`comm-delete` on page 596

`comm` on page 593

`export` statements on page 623

`import` statements on page 605

ext-comm

Name

ext-comm - specifies a set of extended communities to match

Syntax

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

Parameters

extended_community_list - specifies the set of extended communities to be matched

Description

The **ext-comm** clause is used to specify a set of extended communities that must be matched for a route to be considered a match.

Defaults

The default is to ignore the communities attribute.

Context

BGP import statements or BGP subclauses of export statements

Examples

The following example exports to BGP peers in AS 65534 all routes that were learned from BGP peers in AS 65533 with the extended **route-target-as** community 65533 101:

```
export proto bgp as 65534 {  
    proto bgp as 65533 ext-comm { route-target-as 65533 101: } {  
        all;  
    };  
};
```

See Also

"Chapter 30 BGP Communities" on page 133 in *Configuring GateD*

export statements on page 623

ext-comm-add on page 600

ext-comm-delete on page 601

extended_community_list on page 602

import statements on page 605

ext-comm-add

Name

ext-comm-add - specifies a set of extended communities to be added on export

Syntax

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

Parameters

extended_community_list - specifies the set of extended communities to be added to routes matching this export clause

Description

The **ext-comm-add** clause is used to specify a set of extended communities that are to be added to route advertisements matching a particular BGP export statement.

Defaults

The default is to not add extended communities to outgoing route advertisements.

Context

BGP export statements

Examples

The following example adds the **route-target-ip** extended community to BGP routes learned from peers in AS 65533 when they are exported to BGP peers in AS 65534:

```
export proto bgp as 65534 ext-comm-add { route-target-ip 192.0.2.1 100; } {  
    proto bgp as 65533 {  
        all;  
    };  
};
```

See Also

"Chapter 30 BGP Communities" on page 133 in *Configuring GateD*

export statements on page 623

extended_community_list on page 602

ext-comm-delete on page 601

ext-comm on page 599

import statements on page 605

ext-comm-delete

Name

ext-comm-delete - specifies a set of extended communities to be removed on export

Syntax

```
ext-comm-delete { extended_community_list }
```

Parameters

extended_community_list - specifies the set of extended communities to be deleted from routes matching this export clause

Description

The **ext-comm-delete** clause is used to specify a set of communities that are to be removed from routes being advertised as a result of matching the export statement containing the **ext-comm-delete** clause.

Defaults

The default is to not remove extended communities from outgoing route advertisements.

Context

BGP export statements

Examples

The following example removes the **route-origin-as** extended community from all routes learned from BGP peers in AS 65534 when they are advertised to BGP peers in AS 65533.

```
export proto bgp as 65533 ext-comm-delete { route-origin-as 65533 314; } {  
    proto bgp as 65534 {  
        all;  
    };  
};
```

See Also

“Chapter 30 BGP Communities” on page 133 in *Configuring GateD*

export statements on page 623

import statements on page 605

extended_community_list

Name

extended_community_list - a list of extended communities used for matching or modifying AS paths

Syntax

```
route-target-as autonomous_system_number 4_byte_number ;
route-target-ip ip_address 2_byte_number ;
route-origin-as autonomous_system_number 4_byte_number ;
route-origin-ip ip_address 2_byte_number ;
```

Parameters

autonomous_system_number - the autonomous system number of this router

2_byte_number - a positive number in standard GateD number format with a maximum value of $2^{16} - 1$

4_byte_number - a positive number in standard GateD number format with a maximum value of $2^{32} - 1$

Description

An *extended_community_list* specifies a series of extended communities that are to be matched against a route, added to a route on export, or deleted from a route on export. Any number of extended communities can be specified in a single extended community list in any order. No extended community can appear more than once in this list.

Defaults

none

Context

```
ext-comm-add
ext-comm-delete
ext-comm
```

Examples

The following example exports all static routes to BGP peers in AS 65534, adding the **route-origin-ip** extended community with the IP address of 192.0.2.1 and a value of 500:

```
export proto bgp as 65534 ext-comm-add { route-origin-ip 192.0.2.1 500; } {
    proto static {
        all;
    };
};
```

```
};
```

See Also

“Chapter 30 BGP Communities” on page 133 in *Configuring GateD*

`export` statements on page 623

`ext-comm-add` on page 600

`ext-comm-delete` on page 601

`ext-comm` on page 599

`import` statements on page 605

Chapter 27

Route Importation

fromribs

Name

fromribs - specifies the RIBs from which a route will be imported

Syntax

```
import proto ( ... import_parameters ... ) fromribs riblist
    route_filter fromribs riblist ;
```

Parameters

route_filter - a set of route filters specifying routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See “Chapter 28 Route Filtering” on page 129 of *Configuring GateD* for more information.

riblist - one of the following:

unicast - specifies that the routes in the unicast RIB are to be imported

multicast - specifies that the routes in the multicast RIB are to be imported

unicast multicast - specifies that the routes in both the unicast and multicast RIBs are to be imported

Description

fromribs is used to indicate the RIBs from which a route will be imported. It is currently only applicable to BGP because BGP can learn routes in either the unicast or multicast RIBs. When used on a filter, only routes in the specified RIB(s) are matched.

If just one of the unicast or multicast RIBs is specified for **fromribs** and a route to be imported is in both RIBs, the route will be imported to just the specified **fromribs**. This action can be overridden by a **toribs** specification.

Defaults

If **fromribs** is not specified, routes from either RIB are accepted and imported according to **toribs**. If **toribs** is not specified, the route is imported into the RIB, or RIBs from which it came.

Context

`import` statement

route_filter

Examples

```
import proto bgp as 65534 {  
    all fromribs multicast ;  
} ;
```

This example only imports multicast routes learned from this AS.

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

import proto bgp

Name

import proto bgp - specifies BGP import policy

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

Parameters

as *ASN* - specifies that this import statement applies to routes learned from peers in AS *ASN*

comm { *communities_list* } - specifies that this import statement applies to routes learned with the communities specified in *communities_list*

ext-comm { *extended_communities_list* } - specifies that this import statement applies to routes learned with the extended communities specified in *extended_communities_list*

toribs - specifies RIBs into which routes are to be imported

fromribs - specifies RIBs from which routes are to be imported

preference *preference* - specifies the preference value to be assigned to routes that match this import statement or *route_filter*

route_filter - a set of route filters specifying particular routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See “Chapter 28 Route Filtering” on page 129 of *Configuring GateD* for more information.

restrict - specifies that routes matching this import statement or *route_filter* are not to be considered in the route selection process

aspath *aspath-regular-expression* - specifies that this import statement or *route_filter* applies only to routes with an AS_PATH path attribute that matches *aspath-regular-expression*

origin any - specifies that this import statement applies to routes regardless of their ORIGIN path attribute

origin igp - specifies that this import statement applies to routes with an ORIGIN path attribute of IGP

origin egp - specifies that this import statement applies to routes with an ORIGIN path attribute of EGP

origin incomplete - specifies that this import statement applies to routes with an ORIGIN path attribute of incomplete

Description

Import statements specify the policy applied to routes being learned from a particular protocol. They specify what attributes are to be associated with routes when they are placed in the routing table, and the priority the routes are to be given in the route selection process.

BGP supports propagation control by the use of an AS path regular expression, which is documented in “Chapter 29 Matching AS Paths” on page 131 in *Configuring GateD*.

The **comm** option allows the specification of import policy based on the communities attributes found in the BGP update. If multiple communities are specified in the **comm** option, only updates carrying all of the specified communities will be matched. The **ext-comm** option allows the specification of import policy based on the communities attributes found in the BGP update. If multiple communities are specified in the **ext-comm** option, only updates carrying all of the specified communities will be matched. See “Chapter 30 BGP Communities” on page 133 in *Configuring GateD* for more information about communities.

Note that it is quite possible for several BGP import clauses to match a given update. If more than one clause matches, the first matching clause will be used. All later matching clauses will be ignored. For this reason, it is generally desirable to order import clauses from most to least specific. An import clause without a **comm** or **ext-comm** option will match any update regardless of the presence or absence of communities.

BGP stores any routes that were rejected implicitly by not being mentioned in a route filter, or explicitly with the **restrict** keyword, in the routing table with a negative preference. A negative preference prevents a route from becoming active, which prevents it from being installed in the forwarding table or exported to other protocols. This restriction mechanism alleviates the need to terminate and re-establish a session upon reconfiguration if importation policy is changed.

Defaults

By default, all routes from active protocols are imported with default preference.

Context

control statements

Examples

Example 1

The following example imports all routes learned from BGP peers in as 65534.

```
import proto bgp as 65534 {  
    all;  
};
```

Example 2

The following example imports all routes learned from BGP peers regardless of their AS:

```
import proto bgp aspath "(.*)" {  
    all;  
};
```

Example 3

The following example imports all routes learned from BGP peers with a community value of 65534 1000:

```
import proto bgp aspath "(.*)" comm { comm-split 65534 1000 } {  
    all;  
};
```

See Also

"Chapter 30 BGP Communities" on page 133 in *Configuring GateD*

"Chapter 31 Route Importation" on page 137 in *Configuring GateD*

"Communities" on page 77 in *Configuring GateD*

bgp on page 232

preference on page 618

import proto ospfase

Name

import proto ospfase - specifies OSPF import policy

Syntax

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

Parameters

route_filter - a set of route filters specifying particular routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See "Chapter 28 Route Filtering" on page 129 of *Configuring GateD* for more information.

tag tagvalue - specifies that this import statement applies to routes learned with the tag *tagvalue*

preference preference - specifies the preference value to be assigned to routes that match this import statement or *route_filter*

restrict - specifies that routes which match this import statement or *route_filter* are not to be considered during the route selection process

Description

Import statements specify import policy. Import policy determines which, and in what manner, routes are to be added to GateD's internal routing table.

Due to the nature of OSPF, only the importation of ASE routes may be controlled. OSPF intra- and inter-area routes are always imported into the GateD routing table. If a tag is specified, the **import** clause will only apply to routes with the specified tag.

It is only possible to restrict the importation of OSPF ASE routes when a router is functioning as an AS border router. This is accomplished by specifying an **export ospfase** clause. Specification of an empty **export** clause can be used to restrict importation of ASEs when no ASEs are being exported. (For more information about exporting ASEs, see "Exporting to OSPF ASE and NSSA" on page 151 of *Configuring GateD*.)

Like the other interior gateway protocols, **preference** cannot be used to choose between OSPF ASE routes. That is done by the OSPF costs. Routes that are rejected by policy are stored in the table with a negative preference.

Defaults

```
import proto ospfase {  
    all ;  
} ;
```

Context

control statements

Examples

The following example causes no ASE routes learned via the OSPF protocol to be considered during the route selection process.

```
import proto ospfase restrict;
```

The following example causes only ASE routes learned via the OSPF protocol with tag 123 to be imported.

```
import proto ospfase tag 123 {  
    all;  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

ospf on page 124

preference on page 618

import proto redirect

Name

import proto redirect- specifies ICMP redirect import policy

Syntax

```
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 ;
```

Parameters

route_filter - a set of route filters specifying particular routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See "Chapter 28 Route Filtering" on page 129 of *Configuring GateD* for more information.

interface *interface_list* - specifies a list of interfaces to which this import policy applies. This import statement will be applied only to routes learned over the listed interfaces.

gateway *gateway_list* - specifies a list of gateways to which this import policy applies. This import statement will be applied only to routes learned via the listed gateways.

preference *preference* - specifies the preference value to be assigned to routes that match this import statement or *route_filter*

restrict - specifies that routes matching this import statement or *route_filter* are not to be considered during the route selection process

toribs *riblist* - specifies RIBs to import into

Description

Import statements specify import policy. That is, they specify the policy which is used to determine whether and how routes are to be added to the GateD internal routing table.

The importation of Redirect routes can be controlled by any source interface or source gateway.

Context

control statements

Examples

The following example installs redirects for more specific routes of 10/8 only.

```
import proto redirect {  
    10.0.0.0/8 refines ;  
} ;
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

preference on page 618

redirect on page 319

rip on page 69

import proto rip

Name

import proto rip - specifies RIP import policy

Syntax

```
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 ;
```

Parameters

route_filter - a set of route filters specifying particular routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See “Chapter 28 Route Filtering” on page 129 of *Configuring GateD* for more information.

interface *interface_list* - specifies a list of interfaces to which this import policy applies. This import statement will be applied only to routes learned over the listed interfaces.

gateway *gateway_list* - specifies a list of gateways to which this import policy applies. This import statement will be applied only to routes learned via the listed gateways.

preference *preference* - specifies the preference value to be assigned to routes that match this import statement or *route_filter*

restrict - specifies that routes matching this import statement or *route_filter* are not to be considered during the route selection process

toribs *riblist* - specifies RIBs to import into

tag *tagvalue* - specifies the rip route tag value to which this import statement applies

Description

Import statements specify import policy. That is, they specify the policy which is used to determine whether and how routes are to be added to the GateD internal routing table.

The importation of RIP routes can be controlled by any rip tag value, source interface, or source gateway.

Context

control statements

Examples

Example 1

The following example imports, from RIP, all routes to the 10/8 network or any of its sub-nets. It restricts all routes to the 192.168/16 network, and it imports, with a preference of 110, all other routes learned via RIP:

```
import proto rip {  
    10/8 ;  
    192.168/16 exact restrict ;  
    all preference 110 ;  
} ;
```

Example 2

The following example imports, from RIP, all routes learned over the fxp0 interface:

```
import proto rip interface fxp0 {  
    all;  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

preference on page 618

redirect on page 319

rip on page 69

import proto ripng

Name

import proto ripng - specifies RIPng import policy

Syntax

```
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 ;
```

Parameters

route_filter - a set of route filters specifying particular routes to match. These filters will specify whether a route is to be accepted, and if so, with what attributes. See “Chapter 28 Route Filtering” on page 129 of *Configuring GateD* for more information.

interface interface_list - specifies a list of interfaces to which this import policy applies. This import statement will be applied only to routes learned over the listed interfaces.

gateway gateway_list - specifies a list of gateways to which this import policy applies. This import statement will be applied only to routes learned via the listed gateways.

preference preference - specifies the preference value to be assigned to routes that match this import statement or *route_filter*

restrict - specifies that routes matching this import statement or *route_filter* are not to be considered in the selection process

toribs riblist - specifies RIBs to import into

tag tagvalue - specifies that this import statement applies to routes learned with the tag *tagvalue*

Description

This statement configures import policy for RIPng. The configuration syntax and options are very similar to those used by RIP.

Context

control statements

Examples

The following example imports, from RIPng, all routes learned over the fxp0 interface.

```
import proto ripng interface fxp0 {  
    all ;  
} ;
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*
preference on page 618

preference

Name

preference - specifies the preference assigned to a set of routes

Syntax

preference *preference*

Parameters

preference *preference* - specifies the preference value used when comparing this route to other routes from other protocols. The route to a given destination with the lowest preference at any given time becomes the active route. The active route is installed in the forwarding table and is eligible to be exported to other protocols. The default preferences are configured by the individual protocols.

Description

Preference statements are used to assign a preference value to routes. The preference value is the first tie-breaker when determining which route to a given destination will be used. Values must be in the range 0-255.

Defaults

protocol dependent

Context

import statements

route_filter

Examples

The following example causes routes to the 10/8 network, or any of its subnets, to be installed with a preference of 100, and all other routes learned from RIP to be installed with the RIP default preference:

```
import proto rip {  
    10/8 preference 100;  
    all;  
};
```

See Also

"Chapter 31 Route Importation" on page 137 in *Configuring GateD*

import proto bgp on page 607

import proto ospf on page 610

import proto rip on page 614

restrict

Name

`restrict` - specifies that the routes are not desired in the routing table

Syntax

`restrict`

Parameters

none

Description

`restrict` specifies that the routes are not desired in the routing table. In some cases, the routes are not installed in the routing table. In other cases, the routes are installed with a negative preference. This prevents them from becoming active so they will not be installed in the forwarding table or exported to other protocols.

`restrict` can be specified on an `import proto` statement, in which case all routes matched by this import statement will be restricted. It can also be specified in `route_filter`, in which case only routes matching the `route_filter` will be restricted.

Defaults

none

Context

`import` statement

`route_filter`

Examples

The following example causes route learned from RIP to not be considered for route selection:

```
import proto rip restrict;
```

See Also

"Chapter 31 Route Importation" on page 137 in *Configuring GateD*

`import proto bgp` on page 607

`import proto ospfase` on page 610

`import proto rip` on page 614

tag

Name

tag - specifies the protocol-specific tag value to which an **import proto** statement applies

Syntax

```
tag tagvalue
```

Parameters

tagvalue - an integer from 0 to 65535 for RIP and RIPng routes and 0 to 4294967295 for OSPF routes.

Description

Some protocols (RIP, RIPng, and OSPF) allow the setting of a tag value that can be associated with an advertised route. This tag can then be used as an import filter for incoming routes. **tag** specifies that this import statement applies to routes learned with a matching tag. RIP and RIPng allow 16-bit tags, and OSPF allows 32-bit tags to be advertised with routes. Policy configured with **tag** applies only to routes matching the given tag.

Defaults

none

Context

```
import proto rip statement  
import proto ripng statement  
import proto ospfase statement
```

Examples

```
import proto rip tag 100 {  
    all ;  
} ;
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

import proto ospfase on page 610

import proto rip on page 614

import proto ripng on page 616

toribs

Name

toribs - specifies the ribs into which a route should be imported

Syntax

```
import proto ( ... import_parameters ... ) toribs riblist
route_filter toribs riblist ;
```

Parameters

import_parameters - any other import parameters as specified in a given **import** command.

route_filter - a set of route filters specifying particular routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See "Chapter 28 Route Filtering" on page 129 of *Configuring GateD* for more information.

riblist - one of the following:

unicast - specifies that the routes are to be imported into the unicast RIB

multicast - specifies that the routes are to be imported into the multicast RIB

unicast multicast - specifies that the routes are to be imported into both the unicast and multicast RIBs

Description

These parameters are used to specify into which ribs a route should be imported. This is useful when you are running PIM-SM, which does not maintain its own routing table. Instead, it relies on a unicast routing protocol to generate the multicast routing table, which it will use for performing reverse path forwarding.

toribs specified on a matching *route_filter* overrides any **toribs** specified on the **import proto** statement.

Defaults

The default varies based on protocol. MPBGP learns rib information along with routes, and places each route into the appropriate ribs. All other import protocols place routes into the unicast routing table only, by default.

Context

import statement

route_filter

Examples

Example 1

The following example imports all routes learned from RIP in to both the unicast rib and the multicast rib.

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

Example 2

The following example does the same thing, but uses the *route_filter* version.

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

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`import proto bgp` on page 607

`import proto ospfase` on page 610

`import proto rip` on page 614

Chapter 28

Route Exportation

export proto bgp

Name

export proto bgp - specifies the BGP peers that will receive routes

Syntax

```
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
    } ;
```

Parameters

autonomous_system - autonomous system number from 1 to 65535

comm-add { communities_list } - communities to be added. See “Chapter 30 BGP Communities” on page 133 in *Configuring GateD* for more information. By default no communities are added.

comm-delete { communities_list } - communities to be deleted. See “Chapter 30 BGP Communities” on page 133 in *Configuring GateD* for more information. By default no communities are deleted.

ext-comm-add { extended_communities_list } - extended communities to be added. See “Chapter 30 BGP Communities” on page 133 in *Configuring GateD* for more information. By default no extended communities are added.

ext-comm-del { extended_communities_list } - extended communities to be deleted. See “Chapter 30 BGP Communities” on page 133 in *Configuring GateD* for more information. By default no extended communities are deleted.

metric - BGP Multi-exit discriminator metric from 0 to 65535, by default the BGP **metricout** value is used.

export_source_statements - zero or more source statements: **proto bgp**, **proto rip**, **proto ripng**, **proto ospf**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, Or **proto aggregate**.

Description

The **proto bgp** export target statement specifies the neighboring AS to which routes will be distributed for those routes that match the associated *export_source_statements*. Additionally, the **proto bgp** command can be used to set community path attributes through the **comm-add**, **comm-delete**, **ext-comm-add**, and **ext-comm-del** commands.

Like other *export_target_statements*, **proto bgp** allows one to set a metric on the propagated routes. Of course, BGP does not have a metric in the sense that the link state protocols do, instead using a complicated scheme for selecting a route. (Refer to "Route Selection" on page 65 in *Configuring GateD* for more information.) Setting the metric with **proto bgp** sets the MED used in steps 6 and 7 of this process. More detail on MEDs can be found in "Multi-Exit Discriminator Overview and Examples" on page 79 in *Configuring GateD*. From a scoping perspective, metric as set by this command is tighter scope than that set in either the group or peer statement, and looser scope than that set in an *export_source_statement*.

Defaults

BGP will, by default, export all default routes if there is no explicit export policy that applies to this peer.

Context

global statement

Examples

Example 1

Configure GateD to export all EBGp routes to its IBGP peers, where *myAS* is my local AS number.

```
export proto bgp as myAS {  
    proto bgp aspath "(.*)" origin any {  
        all;  
    };  
};
```

Example 2

The following example exports all routes learned via RIP and its external BGP peers to all BGP peers in AS 65534, adds to the advertisement the well-known community NO_EXPORT_SUBCONFED, and strips the arbitrary community 0x123 0x321.

```
export proto bgp as 65534  
    comm-add { community no-export-subconfed; }  
    comm-delete { comm-hex 0x123 0x321; } {
```

```

        proto rip {
            all;
        } ;
        proto bgp aspath "(.*)" origin any {
            all;
        };
    } ;

```

Example 3

The following example exports all OSPF routes to BGP peers in AS 65533.

```

export proto bgp as 65533 {
    proto ospf {
        all;
    };
};

```

Example 4

The following example exports into BGP, to all peers in AS 65533, all, and only, routes generated via an aggregate statement.

```

export proto bgp as 65533 {
    proto aggregate {
        all;
    };
};

```

Example 5

The following example exports into BGP, to all peers in AS 65533, all, and only, routes learned with a leading AS of 65532.

```

export proto bgp as 65533 {
    proto bgp aspath "( 65532 .* )" origin any {
        all;
    };
};

```

See Also

Application of the Border Gateway Protocol in the Internet (RFC 1164) at <http://ietf.org/rfc/rfc1164.txt>

"AS Path Regular Expressions" on page 131 in *Configuring GateD*

"Chapter 31 Route Importation" on page 137 in *Configuring GateD*

comm-add on page 595

`comm-delete` on page 596
`export proto isis` on page 627
`export proto ospfase` on page 630
`export proto ospfnssa` on page 633
`export proto rip` on page 635
`export proto ripng` on page 638
`proto aggregate` on page 643
`proto bgp` on page 646
`proto direct` on page 649
`proto isis` on page 652
`proto kernel` on page 655
`proto ospf` on page 657
`proto ospfase` on page 660
`proto rip` on page 663
`proto static` on page 669

export proto isis

Name

export proto isis - specifies how to inject routes into IS-IS

Syntax

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

Parameters

type - either **internal** or **external**

level - the IS-IS level, either 1 or 2. (Defaults to 2.)

metric - an IS-IS metric from 1 to 63

export_source_statements - zero or more source statements: **proto bgp**, **proto rip**, **proto ospf**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, or **proto aggregate**.

Description

The **proto isis** export target specifies that routes matching the subsequent export sources should be exported via the IS-IS protocol. Normally, only networks associated with interfaces on which IS-IS is being run will be advertised. In order to override this behavior (for example, to advertise directly connected networks on which IS-IS is not being run, or to distribute routes learned from another protocol via IS-IS), a **proto isis** export target statement must be used.

As with other *export_target_statements*, the arguments associated with **proto isis** are specific to the IS-IS protocol, and a clear understanding of the protocol is important in order to implement appropriate policy. For example, explaining the difference between level 1 and level 2 routers is beyond the scope of this document.

The **metric-type** keyword is used to specify whether the metric for routes exported to IS-IS via the export source statement associated with this export target are directly comparable to normal IS-IS metrics. Note that the "wide" metric TLV does not support the marking of external metrics. If this type of reachability is originated the metric will be "internal" regardless of the setting here.

If the metric is to be considered comparable and used in the IS-IS SPF algorithm, then **internal** should be specified. If the metrics are incompatible and should not be used, then **external** should be specified. If no **metric-type** keyword is issued, then the behavior will be that specified by the export-defaults metric-type statement.

It is worth noting that the IS-IS concept of internal and external metrics is not directly analogous to the OSPF concept of type 1 and type 2 ASEs. Specifically, IS-IS has internal and external reachability, rather than the concept of ASE. External reachability (and the associated metric for routes which are externally reachable) is in behavior the same as if

an OSPF route were learned as a type 2 ASE route. However, internal reachability and the associated metrics are directly related to IS-IS routes. (For example, they are not less preferred than IS-IS routes, whereas type 1 ASE routes in OSPF are less preferred than OSPF routes.)

The `level` keyword is used to override the level specified by the `export-defaults level` command. Because this is an overriding operation, if it is desired to export routes both to level 1 and level 2 routers, two export commands will be required with differing `proto isis` export targets and matching export sources.

Finally, the `metric` keyword is used to set the cost on routes matching the subsequent export sources. The exact use of that metric depends on whether the routes are being exported as internal or external. If they are being exported as external, then any metric set is used as is, without adding any IS-IS link costs. This allows one, for example, to specify a certain ASBR for all traffic to matching destinations, regardless of its location within the IS-IS mesh. For internal routes, the behavior is identical to what would happen if it were an IS-IS route from the beginning.

Defaults

By default, nothing is exported into IS-IS.

Context

global statement

Examples

Example 1

The following example injects all BGP routes learned via peers in AS 65534 into IS-IS with external reachability.

```
export proto isis metric-type external {
    proto bgp as 65534 {
        all;
    };
};
```

Example 2

In order to learn the best route to all static routes configured on IS-IS routers, the following can be used.

```
export proto isis metric-type internal {
    proto static {
        all;
    };
};
```

Example 3

The following example configures GateD to export the 1::/64 static route into ISIS IPv6 external reachability

```
export proto isis metric-type external {  
    proto static {  
        1::/64;  
    } ;  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export-defaults` on page 171

`export proto bgp` on page 623

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`export proto ripng` on page 638

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

export proto ospfase

Name

export proto ospfase - specifies how to inject routes into OSPF as ASE

Syntax

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

Parameters

type - the type of AS External route, 1 or 2

tagvalue - an arbitrary number from 0 to 4294967295

metric - OSPF ASE cost from 0 to 16777215

export_source_statements - zero or more source statements: **proto bgp**, **proto rip**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, or **proto aggregate**.

Description

The **proto ospfase** export target specifies that routes matching the subsequent export sources should be exported through the OSPF protocol as AS external (ASE) routes. OSPF uses four different types of routes: intra-area, inter-area, type 1 ASE, and type 2 ASE. These four types of routes are listed in the order in which they are preferred, so if, for example, a route to the same destination is learned both via type 1 ASE and type 2 ASE, then the route learned via type 1 will always be used. 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 (via the *type* option) 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 metric associated with the ASE route is used, and the internal OSPF cost to the ASBR is ignored (except for tie breaking). If no *type* is specified, then the default is to use whatever was configured by the **type** command.

The **tag** command sets an arbitrary number in the tag field for use in policy matching. This tag has no meaning in the context of the OSPF protocol, but is instead used as a filter for matching routes via the various *export_source_statements*.

Finally, the **metric** keyword is used to set the cost on routes matching the subsequent export sources. The exact use of that metric depends on whether the routes are being exported as type 1 or type 2 ASE. If they are being exported as type 2, then any metric set is propagated unchanged throughout the OSPF network, allowing one, for example, to specify that all ASE traffic go through a single ASBR, regardless of that ASBR's location in

the OSPF cloud. If no metric is specified, then the metric will be set according to the `inherit-metric` command.

It is important to note that this only controls the distribution of routes as type 5 LSAs; for distributing routes via type 7 LSAs, separate `proto ospfnssa` export targets must explicitly be configured. Type-5 LSAs (AS External LSAs) are distributed throughout type-5 capable areas in an OSPF domain. They represent destinations external to the AS. Type-7 LSAs are distributed within Not So Stubby Areas (NSSAs) and may be translated or aggregated to type-5 LSAs by the Area Border Routers (ABRs) bordering NSSA area(s).

Defaults

By default, nothing is exported into OSPF ASE.

Context

global statement

Examples

Example 1

The following example injects all BGP routes learned via peers in AS 65534 in to OSPF ASE as type 2 routes. This is not recommended.

```
export proto ospfase type 2 {
    proto bgp as 65534 {
        all;
    };
};
```

Example 2

The following example ensures that all directly connected networks are distributed via OSPF.

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

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`inherit-metric` on page 109
“OSPF Overview” on page 45 in *Configuring GateD*
`proto aggregate` on page 643
`proto bgp` on page 646
`proto direct` on page 649
`proto isis` on page 652
`proto kernel` on page 655
`proto ospf` on page 657
`proto ospfase` on page 660
`proto rip` on page 663
`export proto ripng` on page 638
`proto static` on page 669
`type` on page 150
`stubnetworks` on page 141

export proto ospfnssa

Name

export proto ospfnssa - specifies how to inject ASE routes as type 7 LSAs

Syntax

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

Parameters

type - the type of AS External route, 1 or 2

tagvalue - an arbitrary number from 0 to 4294967295

metric - OSPF ASE cost from 0 to 16777215

export_source_statements - zero or more source statements: **proto bgp**, **proto rip**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, Or **proto aggregate**.

Description

The **proto ospfnssa** export target specifies that routes matching the subsequent export sources should be exported via type 7 LSAs in OSPF. In short, this means that the behavior of this export target is identical to the behavior of the **proto ospfase** target, with the exception that the routes matching subsequent *export_source_statements* will only be advertised to areas that are configured to be "not so stubby."

Defaults

By default, nothing is exported via type 7 LSAs.

Context

global statement

Examples

The following example injects all BGP routes learned via peers in AS 65534 into OSPF ASE as type 2 routes, but only into those areas which are configured as NSSA.

```
export proto ospfnssa type 2 {
    proto bgp as 65534 {
        all;
    };
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto rip` on page 635

`export proto ripng` on page 638

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

export proto rip

Name

export proto rip - specifies how to inject routes into RIP

Syntax

```
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
    };
```

Parameters

interface_list - the list of interfaces over which routes from a subsequent export source will be sent or **all** for all interfaces on which RIP is configured

gateway_list - the list of next hop RIP routers to which routes from a subsequent export source will be sent

tagvalue - an arbitrary number from 0 to 65535

metric - RIP metric number from 1 to 15

export_source_statements - zero or more source statements: **proto bgp**, **proto rip**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, or **proto aggregate**.

Description

The **proto rip** export target specifies how routes are to be readvertised as RIP routes. The **interface** and **gateway** parameters specify that the routes matched in subsequent *export_source_statements* will only be advertised out the associated *interface_list* or *gateway_list*. In the absence of the **interface** or **gateway** keywords, routes will be sent out all interfaces on which RIP is configured.

Because RIP is a broadcast protocol (or multicast in the case of RIPv2), the effect of the **gateway** keyword can be somewhat counter-intuitive. Specifically, the user would expect RIP routes to only be advertised to the specific gateways mentioned in an instance of the **proto rip** export target statement. This is in fact the case if those gateways are source gateways. However, if source gateways are not used, then **gateway** effectively acts as a wildcard for **interface**. By specifying gateways, the routes which pass the export source filter associated with an export command will be either broadcast or multicast (as appropriate) to all RIP routers on the interfaces whose subnets include the specified gateways.

It is important to note that the *gateway_list* must only include gateways on directly connected interfaces; otherwise a parse error will occur. If the interface upon which a specified gateway is directly reachable may not be functional at the time GateD configuration takes place, then the **interfaces define** statement should be used to predefine the interface.

Finally, the **metric** keyword is used to set the RIP metric on routes matching the subsequent export sources. This keyword is used to override the RIP metric set with the **rip defaultmetric** command.

Defaults

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

Context

global statement

Examples

Example 1

The following example exports all, and only, routes specified in the static clause of the GateD configuration file via RIP.

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

Example 2

The following example advertises all static, direct, and RIP routes, via RIP, out the interface 192.168.2.10.

```
export proto rip interface 192.168.2.10 {  
    proto static {  
        all;  
    };  
    proto direct {  
        all;  
    };  
    proto rip {
```

```
        all;  
    };  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`defaultmetric` on page 55

`define` on page 22

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`export proto ripng` on page 638

`proto static` on page 669

`sourcegateways` on page 76

`version` on page 83

export proto ripng

Name

export proto ripng - specifies how to inject routes into RIPng

Syntax

```
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
    };
```

Parameters

interface_list - the list of interfaces over which routes from a subsequent export source will be sent or **all** for all interfaces on which RIPng is configured

gateway_list - the list of next hop RIPng routers to which routes from a subsequent export source will be sent

tagvalue - an arbitrary number from 0 to 65535

metric - RIPng metric number from 1 to 15

export_source_statements - zero or more source statements: **proto bgp**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, or **proto aggregate**.

Description

The **proto ripng** export target specifies how IPv6 routes are to be readvertised as RIPng routes. The interface and gateway parameters specify that the routes matched in subsequent *export_source_statements* will only be advertised out the associated *interface_list* or *gateway_list*. In the absence of the **interface** or **gateway** keywords, routes will be sent out all interfaces on which RIPng is configured.

Because RIPng uses multicast, the effect of the gateway keyword can be somewhat counter-intuitive. Specifically, the user would expect RIPng routes to only be advertised to the specific gateways mentioned in an instance of the **proto ripng** export target statement. This is in fact the case if those gateways are source gateways. However, if source gateways are not used, then **gateway** effectively acts as a wildcard for **interface**. By specifying gateways, the routes which pass the export source filter associated with an export command will be either broadcast or multicast (as appropriate) to all RIPng routers on the interfaces whose subnets include the specified gateways.

It is important to note that the *gateway_list* must only include gateways on directly connected interfaces; otherwise a parse error will occur. If the interface upon which a speci-

fied gateway is directly reachable may not be functional at the time GateD configuration takes place, then the **interfaces define** statement should be used to predefine the interface.

Finally, the **metric** keyword is used to set the RIPng metric on routes matching the subsequent export sources. This keyword is used to override the RIPng metric set with the **ripng defaultmetric** command.

Defaults

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

Context

global statement

Examples

Example 1

The following example exports all, and only, IPv6 routes specified in the static clause of the GateD configuration file via RIPng.

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

Example 2

The following example advertises all static, direct, and RIPng IPv6 routes, via RIPng, out the interface fec0::01.

```
export proto ripng interface fec0::01 {
    proto static {
        all;
    };
    proto direct {
        all;
    };
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`defaultmetric` on page 557

`define` on page 22

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

`sourcegateways` on page 76

`version` on page 83

fromribs

Name

fromribs - specifies the RIBs from which a route will be exported

Syntax

```
route_filter fromribs riblist ;
```

Parameters

route_filter - a set of route filters specifying routes to match. These filters specify whether a route is to be accepted, and if so, with what attributes. See "Chapter 28 Route Filtering" on page 129 of *Configuring GateD* for more information.

riblist - one of the following:

unicast - specifies that the routes in the unicast RIB are to be exported

multicast - specifies that the routes in the multicast RIB are to be exported

unicast multicast - specifies that the routes in both the unicast and multicast RIBs are to be exported

Description

fromribs is used to indicate the RIBs from which a route will be exported. It is currently only applicable to *route_filters* within a BGP export target because only BGP can export routes from either the unicast or multicast RIBs. It should be used only on **route_filters** associated with the **bgp**, **direct**, or **static** export source statements, as only these protocols can import routes in the multicast RIB.

If just one of the unicast or multicast RIBs is specified for **fromribs** and a route to be exported is in both RIBs, the route will be exported to just the specified **fromribs**.

Defaults

If **fromribs** is not specified, routes from either RIB are accepted and exported according to **toribs**. By default, BGP exports routes in either the unicast or multicast RIBs. All other export protocols export only routes in the unicast RIB.

Context

```
route_filter
```

Examples

```
export proto bgp as 65534 {
    all fromribs multicast ;
} ;
```

This example exports only multicast routes learned from this AS.

See Also

“Chapter 32 Route Exportation” on page 145 in *Configuring GateD*

`proto bgp` on page 646

`proto direct` on page 649

`proto static` on page 669

proto aggregate

Name

proto aggregate - filters on routes which are aggregates of other routes

Syntax

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

Parameters

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filter*s. It indicates that any matching routes are not to be exported.

Description

The **proto aggregate** export source statement is used to match routes which have been aggregated from other routes for exportation via one of the export target statements, described in this chapter.

In the **restrict** version of this *export_source_statement*, any aggregated routes are prohibited from redistribution in the associated *export_target_statement*.

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target. Upon first inspection, this may seem like a non-sequitur as we are matching aggregate routes, but in fact it is not. Keep in mind that an aggregated route is distinct from one which contributes to an aggregate, and also that it is possible to aggregate from other, previously aggregated routes.

As with all **export** source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the *export_target_statement*. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters will be passed on to the export target within which this **proto aggregate** is included.

Any number of unique **proto aggregate** export sources can be used within the context of a single **export** target statement.

Defaults

By default, no `proto aggregate` export source exists.

Context

`export` source statement

Examples

Example 1

The following example exports into OSPF as ASE all routes which are aggregates of other routes.

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

Example 2

The following example exports into RIP all routes which are aggregates of other routes, provided that they themselves are not contributing to a greater aggregate.

```
export proto rip {  
    proto aggregate noagg {  
        all;  
    };  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto bgp

Name

proto bgp - filters on routes learned via BGP

Syntax

```
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 ) ] ; ] } ;
```

Parameters

autonomous_system - autonomous system number from 1 to 65535

aspath_regular_expression - The syntax of these regular expressions is described in "AS Path Regular Expressions" on page 131, and in section 4.2 of RFC 1164.

comm { *communities_list* } - specifies that this import statement applies to routes learned with the communities specified in *communities_list*

ext-comm { *extended_communities_list* } - specifies that this import statement applies to routes learned with the extended communities specified in *extended_communities_list*

metric - a metric with a range appropriate to the protocol as described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

fromribs *riblist* - specifies the RIBs from which routes matching this *route_filter* will be exported. See "**fromribs**" on page 641 for more information.

Description

The `proto bgp` export source statement is used to match routes learned via the BGP protocol for exportation via one of the export target statements, described in this chapter.

The statement includes either an `as` or `aspath` component, an optional community matching component, an optional extended community component, and an optional `noagg` component. If any `route_filters` are provided, then any routes matching those filters which also match the `as` portion and the optional community portions will be passed onto the export target for which this `proto bgp` is an export source. The `noagg` matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

Any number of `proto bgp` export sources can be used within the context of a given export target, so long as they are not repeated.

As with all export source statements, the optional `metric` must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any `route_filters` are provided, then any routes matching these filters that also match the `proto bgp` statement parameters will be passed on to the export target within which this `proto bgp` statement is included.

Defaults

By default there are no `proto bgp` export source statements.

Context

`export` source statement

Examples

Example 1

The following example exports into BGP, to all peers in AS 65533, all, and only, routes learned with a leading AS of 65532.

```
export proto bgp as 65533 {
    proto bgp aspath "( 65532 .* )" origin any {
        all;
    };
};
```

Example 2

The following example injects all BGP routes learned via peers in AS 65534 in to OSPF ASE as type 2 routes. This is not recommended.

```
export proto ospfase type 2 {
    proto bgp as 65534 {
        all;
    };
};
```

```
    };  
};
```

Example 3

The following example exports into BGP, to all peers in AS 65533, all, and only, routes learned with a leading AS of 65532.

```
export proto bgp as 65533 {  
    proto bgp aspath "( 65532 .* )" origin any {  
        all;  
    };  
};
```

See Also

Application of the Border Gateway Protocol in the Internet (RFC 1164) at <http://ietf.org/rfc/rfc1164.txt>

"AS Path Regular Expressions" on page 131 in *Configuring GateD*

"Chapter 31 Route Importation" on page 137 in *Configuring GateD*

`comm` on page 593

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`fromribs` on page 641

`proto aggregate` on page 643

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto direct

Name

proto direct - filter on directly connected interfaces

Syntax

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

Parameters

interface_list - the list of interfaces on which the matching route must be present in order for it to be exported to the associated export target

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

fromribs riblist - specifies the RIBs from which routes matching this *route_filter* will be exported. See "**fromribs**" on page 641 for more information.

Description

The **proto direct** export source statement is used to match routes associated with directly connected interfaces for exportation via one of the export target statements, described in this chapter.

If the optional interface parameter is used, only routes associated with those interfaces matching the *interface_list* will be tested against the remaining filters and policy.

In the **restrict** version of this export source statement, any direct routes which match the optional interface parameter (or all direct routes in the absence of said parameter) will not be exported via the associated *export_target_statement*.

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target. Refer to "Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD* for more information.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the *export_target_statement*. Setting a metric value in

an export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters that also match the optional interface specification will be passed on to the export target within which this `proto direct` is included.

Any number of unique `proto direct` export sources can be used within the context of a single export target statement.

Defaults

By default, no `proto direct` export source exists (except as described in the `export` command).

Context

`export` source statement

Examples

Example 1

The following example exports into RIP all directly connected routes.

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

Example 2

The following example exports into RIP all directly connected routes, except those on the interface eth0.

```
export proto rip {
    proto direct interface eth0 restrict;
    proto direct {
        all;
    };
};
```

Example 3

The following example exports into RIP all directly connected routes, but routes from interface eth0 are exported with a higher metric. **Note:** The default rip metric is 1.

```
export proto rip {
    proto direct eth0 metric 2 {
        all;
    };
};
```

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

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto isis

Name

proto isis - filters on routes learned via IS-IS

Syntax

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

Parameters

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See “Chapter 28 Route Filtering” on page 129 for more information.

internal - explicitly propagates only those routes which have internal reachability

external - explicitly propagates only those routes which have external reachability

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto isis** export source statement is used to match routes that have been propagated through IS-IS for exportation via one of the export target statements, described in this chapter.

The **internal** and **external** keywords are used to explicitly propagate only those routes which have internal or external reachability, respectively. In the absence of either keyword, the **proto isis** export source statement defaults to only matching IS-IS routes that have been received with internal reachability. As a result, in order to propagate all IS-IS routes, a separate export source must be created for both internal and external reachability.

In the **restrict** version of this *export_source_statement*, no IS-IS routes of the appropriate reachability will be exported via the associated *export_target_statement*. Strictly speaking, using the **restrict** version of this statement has no effect, as it would be the same behavior as if the statement had not been specified. (In other words, no IS-IS routes other than those matching any unrestricted version of the statement would be exported to the associated target.)

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters that also match the **proto isis** statement parameters will be passed on to the export target for which this **proto isis** is an export source.

Any number of unique **proto isis** export sources can be used within the context of a single export target statement.

Defaults

By default, no **proto isis** export source exists.

Context

export source statement

Examples

Example 1

The following example exports into OSPF as Type 1 ASE all IS-IS routes with internal reachability.

```
export proto ospfase type 1 {
    proto isis {
        all;
    };
};
```

Example 2

The following example exports into RIP all IS-IS routes.

```
export proto rip {
    proto isis internal {
        all;
    };
    proto isis external {
        all;
    };
};
```

Example 3

The following example exports into OSPF as ASE type 2 routes, all IS-IS routes with external reachability.

```
export proto ospfase type 2 {
```

```
    proto isis external {  
        all;  
    };  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

“OSPF Overview” on page 45 in *Configuring GateD*

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto kernel

Name

proto kernel - filters on routes that are learned from the FIB

Syntax

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

Parameters

interface_list - the list of interfaces on which the next hop for the matching route must reside in order for the matching route to be exported to the associated export target

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto kernel** export source statement is nearly identical to the **proto static** export source statement. The only difference is that, rather than referring to routes statically configured by the routing daemon, this refers to routes which are learned from the FIB. Of course, it is not normally expected that routes will be learned from the FIB; however, there are two cases where this is possible. First, the routes could be remnant routes learned via the route socket after a software crash. In this case, it is strongly advised that the routes not be readvertised. Second, another administrative authority has directly added routes to the FIB. This is most commonly the case when GateD is used as a routing daemon on a UNIX workstation (in which case the FIB is the kernel RIB) and the super user adds routes via the route command. Because this is the only case where redistributing kernel routes is even remotely advisable, and the case is an historic one, the **proto kernel** export source should probably never be used.

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

Defaults

By default, no **proto kernel** export source exists.

Context

`export` source statement

Examples

The following example exports into OSPF as ASE all routes which are learned from the FIB.

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

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto ospf` on page 657

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto ospf

Name

proto ospf - filter on routes learned via OSPF

Syntax

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

Parameters

type - the type of AS External route, 1 or 2

tagvalue - an arbitrary number from 0 to 4294967295

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto ospf** export source statement is used to match OSPF routes for exportation via one of the export target statements, described in this chapter. It only deals with routes learned directly from OSPF and does not consider routes that are AS external to OSPF. For these routes, see the **proto ospfase** export source statement.

In the **restrict** version of this *export_source_statement*, no OSPF routes that match the optional *tagvalue* (or no OSPF routes at all in the absence of the *tagvalue* parameter) will be exported via the associated *export_target_statement*. Strictly speaking, using the **restrict** version of this statement without the *tagvalue* parameter has no effect, as it would be the same behavior as if the statement had not been specified (that is, no OSPF routes other than those matching the unrestricted version of the statement would be exported to the associated target).

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any `route_filters` are provided, then any routes matching these filters which also match the `proto ospf` statement will be passed on to the export target for which this `proto ospf` is an export source.

Any number of unique `proto ospf` export sources can be used within the context of a single export target statement.

Defaults

By default, no `proto ospf` export source exists.

Context

`export` source statement

Examples

Example 1

The following example exports into RIP all OSPF routes.

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

Example 2

The following example exports into RIP all OSPF routes, except those received with a tag of 12345.

```
export proto rip {
    proto ospf {
        all;
    };
    proto ospf tag 12345 restrict;
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospfase` on page 660

`proto rip` on page 663

`proto static` on page 669

proto ospfase

Name

proto ospfase - filters on routes learned via OSPF that are AS External

Syntax

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

Parameters

type - the type of AS External route, 1 or 2

tagvalue - an arbitrary number from 0 to 4294967295

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto ospfase** export source statement is used to match routes that have been propagated through OSPF as ASE for exportation via one of the export target statements, described in this chapter. To export routes learned as OSPF routes, see **proto ospf**.

In the **restrict** version of this *export_source_statement*, no OSPF ASE routes that match the optional *tagvalue* (or no OSPF routes at all in the absence of the *tagvalue* parameter) and are of the optional *type* will be exported via the associated *export_target_statement*. In the absence of the *type* parameter, this export source applies to both type 1 and type 2 ASE routes. For more information on the difference between type 1 and type 2 ASE routes, refer to "OSPF Overview" on page 45 in *Configuring GateD*. Strictly speaking, using the **restrict** version of this statement with neither the **tag** nor the **type** parameter has no effect, as it would be the same behavior as if the statement had not been specified. (In other words, no OSPF ASE routes other than those matching any unrestricted version of the statement would be exported to the associated target.)

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an

export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters which also match the `proto ospfase` statement parameters will be passed on to the export target within which this `proto ospfase` is included.

Any number of unique `proto ospfase` export sources can be used within the context of a single export target statement.

Defaults

By default, no `proto ospfase` export source exists.

Context

`export` source statement

Examples

Example 1

The following example exports into RIP all OSPF ASE routes.

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

Example 2

The following example exports into RIP all OSPF ASE routes, except those received with a tag of 12345.

```
export proto rip {  
    proto ospfase {  
        all;  
    };  
    proto ospfase tag 12345 restrict;  
};
```

Example 3

The following example exports into RIP all OSPF ASE routes, but type 1 routes are exported with a higher RIP metric.

```
export proto rip {  
    proto ospfase type 2 {  
        all;  
    };  
};
```

```
        proto ospfase type 1 metric 2 {  
            all;  
        };  
    };
```

Example 4

In this example, type 2 OSPF ASE routes that have a tag of 65535 are exported to AS 65535 via BGP, provided that they are not contributing to an aggregate.

```
export proto bgp as 65535 {  
    proto ospfase type 2 tag 65535 noagg {  
        all;  
    };  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto rip` on page 663

`proto static` on page 669

proto rip

Name

proto rip - filter on routes learned via RIP

Syntax

```
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 ) ] ; ] } ;
```

Parameters

interface_list - the list of interfaces over which the matching route must have been received in order for it to be exported to the associated export target

gateway_list - the list of next hop RIP routers from which routes must have been learned in order for them to be exported via the associated export target

tagvalue - an arbitrary number from 0 to 65535

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto rip** export source statement is used to match RIP routes for exportation via one of the export target statements, described in this chapter.

If the optional interface parameter is used, only routes learned over those interfaces matching the *interface_list* will be tested against the remaining filters and policy. Similarly, if the optional gateway parameter is used, then only those routes learned from routers matching the *gateway_list* will be considered for exportation.

In the **restrict** version of this *export_source_statement*, any RIP routes which match the optional **tag**, **interface**, or **gateway** parameters (or all RIP routes in the absence of said parameters) will be excluded from exportation via the associated *export_target_statement*.

The `noagg` matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target. Similarly, the `tag` matching filter is used to specify that any route matching the other filters must also have this administratively set `tagvalue`.

As with all export source statements, the optional `metric` parameter must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any `route_filters` are provided, then any routes matching these filters which also match the `proto rip` statement parameters will be passed on to the export target within which this `proto rip` is included. Note, however, that this export source cannot be used in conjunction with the `proto rip` export target statement. (Technically, exporting from RIP into RIP can be specified, but it has no effect, regardless of additional parameters.)

Any number of unique `proto rip` export sources can be used within the context of a single export target statement.

Defaults

By default, no `proto rip` export source exists.

Context

`export` source statement

Examples

Example 1

The following example exports into OSPF as ASE all RIP routes.

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

Example 2

The following example exports into OSPF as ASE all RIP routes, except that those tagged with 12345 are exported with a higher cost.

```
export proto ospfase {
    proto rip tag 12345 metric 5000 {
        all;
    };
    proto rip {
        all;
    };
};
```

```
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto static` on page 669

proto ripng

Name

proto ripng - filter on routes learned via RIPng

Syntax

```
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 ) ] ; ] } ;
```

Parameters

interface_list - the list of interfaces over which the matching route must have been received in order for it to be exported to the associated export target

gateway_list - the list of next hop RIPng routers from which routes must have been learned in order for them to be exported via the associated export target

tagvalue - an arbitrary number from 0 to 65535

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

Description

The **proto ripng export** source statement is used to match RIPng IPv6 routes for exportation via one of the export target statements, described in this chapter.

If the optional interface parameter is used, only routes learned over those interfaces matching the *interface_list* will be tested against the remaining filters and policy. Similarly, if the optional gateway parameter is used, then only those routes learned from routers matching the *gateway_list* will be considered for exportation.

In the **restrict** version of this *export_source_statement*, any RIPng routes which match the optional **tag**, **interface**, or **gateway** parameters (or all RIPng routes in the absence of said parameters) will be excluded from exportation via the associated *export_target_statement*.

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target. Similarly, the **tag** matching filter is used to specify that any route matching the other filters must also have this administratively set *tagvalue*.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the export target statement. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters which also match the **proto ripng** statement parameters will be passed on to the export target within which this **proto ripng** is included. Note, however, that this export source cannot be used in conjunction with the **proto ripng** export target statement. (Technically, exporting from RIPng into RIPng can be specified, but it has no effect, regardless of additional parameters.)

Any number of unique **proto ripng** export sources can be used within the context of a single export target statement.

Defaults

By default, no **proto ripng** export source exists.

Context

export source statement

Examples

Example 1

The following example exports all RIPng routes to BGP peers in AS 201.

```
export proto bgp as 201 {
    proto ripng {
        all;
    };
};
```

Example 2

The following example exports into IS-IS external reachability all RIPng routes, except that those tagged with 12345 are exported with a higher cost.

```
export proto isis level 2 {
    proto ripng tag 12345 metric 5000 {
        all;
    };
    proto ripng {
        all;
    };
};
```

```
    };  
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

`export proto bgp` on page 623

`export proto isis` on page 627

`export proto ospfase` on page 630

`export proto ospfnssa` on page 633

`export proto rip` on page 635

`proto aggregate` on page 643

`proto bgp` on page 646

`proto direct` on page 649

`proto isis` on page 652

`proto kernel` on page 655

`proto ospf` on page 657

`proto ospfase` on page 660

`proto static` on page 669

proto static

Name

proto static - filter on statically configured routes

Syntax

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

Parameters

interface_list - the list of interfaces on which the next hop for the matching route must reside in order for the matching route to be exported to the associated export target

metric - a metric with range appropriate to the protocol described in the associated export target

route_filter - route filter as described in the route filter document. See "Chapter 28 Route Filtering" on page 129 for more information.

noagg - specifies that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target

restrict - can be specified on the **proto** statement or any of its *route_filters*. It indicates that any matching routes are not to be exported.

fromribs riblist - specifies the RIBs from which routes matching this *route_filter* will be exported. See "**fromribs**" on page 641 for more information.

Description

The **proto static** export source statement is used to match routes which have been statically configured for exportation via one of the export target statements, described in this chapter.

If the optional interface parameter is used, only routes associated with those interfaces matching the *interface_list* will be tested against the remaining filters and policy. Since static routes are not learned, per se, the match against interface may seem slightly different than that of other *export_source_statements* (though it technically is not). The static routes must have next hops that reside on a matched, directly connected interface in order for them to match this filter.

In the **restrict** version of this *export_source_statement*, any static routes that match the optional interface parameter (or all static routes in the absence of said parameter) will not be exported via the associated *export_target_statement*.

The **noagg** matching filter is used to specify that any route matching the other filters has the additional caveat that it must not be contributing to an aggregate in order for it to be passed on to the export target.

As with all export source statements, the optional *metric* parameter must be a valid metric for the associated protocol in the *export_target_statement*. Setting a metric value in an export source overrides any metric set on the export target, thus allowing for additional scope.

If any *route_filters* are provided, then any routes matching these filters which also match the **proto static** statement parameters will be passed on to the export target within which this **proto static** is included.

Any number of unique **proto static** export sources can be used within the context of a single export target statement.

Defaults

By default, no **proto static** export source exists.

Context

export source statement

Examples

Example 1

The following example exports into OSPF as ASE all static routes configured on this router.

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

Example 2

The following example exports into RIP all statically configured routes, except those with next hops on the interface eth0.

```
export proto rip {
    proto static interface eth0 restrict;
    proto static {
        all;
    };
};
```

See Also

“Chapter 31 Route Importation” on page 137 in *Configuring GateD*

export proto bgp on page 623

export proto isis on page 627

export proto ospfase on page 630

`export proto ospfnssa` on page 633
`export proto rip` on page 635
`fromribs` on page 641
`proto aggregate` on page 643
`proto bgp` on page 646
`proto direct` on page 649
`proto isis` on page 652
`proto kernel` on page 655
`proto ospf` on page 657
`proto ospfase` on page 660
`proto rip` on page 663

Chapter 29

Route Aggregation and Generation

aggregate

Name

aggregate - creates a summary (aggregate) route from more specific contributor routes

Syntax

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

Parameters

[**inet6**] **default** - sets the prefix for the aggregate to 0.0.0.0/0 or ::/0 if **inet6** is specified

network - the address of the aggregate, in dotted-quad format (xxx.xxx.xxx.xxx) or IPv6 format

mask - the address mask in dotted-quad format (xxx.xxx.xxx.xxx) or IPv6 format

masklen - the number of contiguous one bits at the beginning of the mask

preference_value - a preference number (integer) from 0 to 255, inclusive

bgp - specifies that this aggregate will use BGP rules to determine whether or not to include each route

brief - specifies that the AS path should be truncated to the longest common AS path

toribs - specifies that the aggregate is restricted to this RIB

aggregate_list - one or more of the following: **proto bgp**, **proto rip**, **proto ripng**, **proto ospf**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, **proto aggregate**, or **proto all** as defined later in this document

Description

Route aggregation is a method of generating a more general route, given the presence of a 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.

Defaults

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.

Context

`global` statement

Examples

Example 1

```
aggregate 10/8 {  
    proto bgp aspath "(64512 .*)" origin any {  
        10/8 refines;  
    };  
};
```

Example 2

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

See Also

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

as

Name

as - restricts selection of routes to those learned from the specified autonomous system

Syntax

as *ASN*

Parameters

ASN - the autonomous system number from which routes are to be learned

Description

as restricts selection of routes to those learned from the specified autonomous system.

Defaults

none

Context

aggregate proto bgp statement

generate proto bgp statement

Examples

```
aggregate 10/8 bgp {  
    proto bgp as 64512 {  
        10/8 refines;  
    };  
};
```

See Also

aggregate on page 673

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

generate on page 678

proto bgp on page 686

aspath

Name

aspath - restricts selection of routes to those that match the specified AS path

Syntax

aspath *aspath-regular-expression*

Parameters

aspath-regular-expression - an AS path regular expression. The syntax of these regular expressions is described in "AS Path Regular Expressions" on page 131, and in Section 4.2 of RFC 1164.

Description

aspath restricts selection of routes to those that match the specified AS path.

Defaults

none

Context

aggregate proto bgp statement

generate proto bgp statement

Examples

```
aggregate 10/8 {  
    proto bgp aspath "(64512 .*)" origin any {  
        10/8 refines;  
    };  
};
```

See Also

aggregate on page 673

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

generate on page 678

proto bgp on page 686

brief

Name

brief - specifies that the AS path should be truncated to the longest common AS path

Syntax

brief

Parameters

none

Description

brief specifies that the AS path should be truncated to the longest common AS path.

Defaults

The default is to build an AS path consisting of SETs and SEQUENCEs of all contributing AS paths.

Context

aggregate statement

Examples

```
aggregate 10/8 bgp brief {  
    proto bgp {  
        10/8 refines;  
    };  
};
```

See Also

aggregate on page 673

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

generate

Name

generate - creates a summary (aggregate) route from one or more specific component routes

Syntax

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

Parameters

[**inet6**] **default** - sets the prefix for the generate to 0.0.0.0/0 or ::/0 if **inet6** is specified

network - the generate address, in dotted-quad format (xxx.xxx.xxx.xxx) or IPv6 format

mask - the address mask (modification) in dotted-quad format (xxx.xxx.xxx.xxx) or IPv6 format

masklen - the number of contiguous one bits at the beginning of the mask

preference_value - a preference number (integer) from 0 to 255, inclusive

noinstall - specifies that the generated route is not installed in the FIB

toribs - specifies that the aggregate is restricted to this RIB

aggregate_list - one or more of the following: **proto bgp**, **proto rip**, **proto ripng**, **proto ospf**, **proto ospfase**, **proto direct**, **proto static**, **proto kernel**, **proto isis**, **proto aggregate**, or **proto all** as defined later in this document

Description

Route generation is a method of generating a more general route, given the presence of a specific route. **generate** differs from **aggregate** only in terms of the route installed in the kernel. **generate** causes a route to the specified generate address to be installed if any contributor route is active. Its primary use is to generate a default route.

Defaults

By default, generate applies to all RIBs to which any contributing route applies. For example, a generate applies to the unicast RIB if and only if any contributing route applies to the unicast RIB.

Context

global statement

Examples

Example 1

```
generate 10/8 {  
    proto bgp aspath "(64512 .*)" origin any {  
        10/8 refines;  
    };  
};
```

Example 2

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

See Also

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

preference

Name

preference - specifies the aggregation preference value to be assigned to a contributor route

Syntax

preference *preference_value*

Parameters

preference_value - a preference number (integer) from 0 to 255, inclusive

Description

preference specifies the aggregation preference for a contributing aggregate route. 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.

Defaults

preference 130;

Context

aggregate statement

generate statement

route_filter

Examples

```
aggregate 10/8 preference 130 bgp {
```

```
    proto bgp {  
        10/8 refines;  
    };  
};
```

See Also

aggregate on page 673

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

generate on page 678

proto aggregate

Name

proto aggregate - filters on routes that are aggregates of other routes

Syntax

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

Parameters

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol- or *route_filter*-specific.

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto aggregate** or *route_filter*

Description

The **proto aggregate** aggregate source statement is used to match routes that have been aggregated from other routes for inclusion in an aggregate route.

In the **restrict** version of this aggregate source statement, any aggregated routes are prohibited from inclusion in the associated aggregation target command.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate for which this **proto aggregate** is an aggregate source. Any number of **proto aggregate** sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto aggregate** export source exists.

Context

aggregate statement

generate statement

Examples

The following example aggregates only certain other aggregates into a more general aggregate.

```
aggregate 10/14 {
```

```
    proto aggregate {  
        10.1/16;  
        10.2/16;  
    } ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto all

Name

proto all - filters on routes learned via any source

Syntax

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

Parameters

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol- or *route_filter*-specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto all** or *route_filter*

Description

The **proto all** aggregate source statement is used to match routes from any source for inclusion in the associated aggregate route. The primary use of this aggregate source statement is to include all potential contributors into an aggregate statement. Protocol-specific aggregate source statements with **restrict** can then be used to limit the set of actual contributors to the aggregate.

In the **restrict** version of this aggregate source statement, no routes from any source will be included in the associated aggregate.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate for which this **proto all** is an aggregate source. Any number of **proto all** sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto all** aggregate source exists.

Context

aggregate statement

generate statement

Examples

This example includes all contributors, except those from kernel or static in the 10/8 aggregate.

```
aggregate 10/8 {  
    proto kernel restrict ;  
    proto static restrict ;  
    proto all {  
        10/8 refines ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto bgp

Name

proto bgp - filters on routes learned via BGP

Syntax

```
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 ;
```

Parameters

ASN - autonomous system number from 1 to 65535

aspath-regular-expression - The syntax of these regular expressions is described in “AS Path Regular Expressions” on page 131, and in Section 4.2 of RFC 1164.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto bgp** or *route_filter*

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See “Chapter 28 Route Filtering” on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol- or *route_filter*-specific.

comm { communities_list } - specifies that this aggregate source statement applies to routes learned with the communities specified in *communities_list*

ext-comm { extended_communities_list } - specifies that this aggregate source statement applies to routes learned with the extended communities specified in *extended_communities_list*

Description

The **proto bgp** aggregate source statement is used to match routes learned via the BGP protocol for inclusion in an aggregate route.

The statement includes an optional **as** or **aspath** component. If any *route_filters* are provided, then any routes matching those filters which also match the **as** portion will be included in the aggregate route for which this **proto bgp** is the source. Any number of

`proto bgp` aggregate sources can be used within the context of a given aggregate statement, so long as they are not repeated. `restrict` specified on a `route_filter` indicates matching routes are not to be contributors.

Defaults

By default, there are no `proto bgp` aggregate source statements.

Context

`aggregate` statement

`generate` statement

Examples

```
aggregate 192.2.0.0 masklen 24 {  
    proto bgp as 65412 {  
        192.2.0.0/24 refines ;  
    } ;  
} ;
```

See Also

Application of the Border Gateway Protocol in the Internet (RFC 1164) at <http://ietf.org/rfc/rfc1164.txt>

"AS Path Regular Expressions" on page 131 in *Configuring GateD*

"Chapter 30 BGP Communities" on page 133 in *Configuring GateD*

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

proto direct

Name

proto direct - filter on directly connected interfaces

Syntax

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

Parameters

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol- or *route_filter*-specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto direct** or *route_filter*

Description

The **proto direct** aggregate source statement is used to match routes associated with directly connected interfaces for inclusion in an aggregate route.

In the **restrict** version of this aggregate source statement, no direct routes will be included in the aggregate in which this **proto direct** occurs.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate for which this **proto direct** is a source. Any number of **proto direct** aggregate sources can be used within the context of a single aggregate command.

restrict specified on a *route_filter* indicates matching routes are not to be contributors.

Defaults

By default, no **proto direct** aggregate source exists.

Context

aggregate statement

generate statement

Examples

The following example includes all direct (interface) addresses that are more specific than 10/16, into a 10/16 aggregate route.

```
aggregate 10/16 {
```

```
    proto direct {  
        10/16 refines ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto isis

Name

proto isis - filters on routes learned via IS-IS

Syntax

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

Parameters

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto isis** or *route_filter*

Description

The **proto isis** aggregate source statement is used to match routes that have been propagated through IS-IS for inclusion in an aggregate route.

In the **restrict** version of this aggregate source statement, no IS-IS routes will be included in the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate for which this **proto isis** is an aggregate source. Any number of **proto isis** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto isis** aggregate source exists.

Context

aggregate statement

generate statement

Examples

The following example aggregates all more specific routes of 10./16 except for 10.1.1.42, if learned from IS-IS.

```
aggregate 10.1/16 {
```

```
    proto all {  
        10.1/16 refines ;  
    } ;  
    proto isis {  
        10.1.1.42 restrict ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto kernel

Name

proto kernel - filters on routes that are learned from the FIB

Syntax

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

Parameters

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto kernel** or *route_filter*

Description

The **proto kernel** aggregate source statement is nearly identical to the **proto static** aggregate source statement. The only difference is that, rather than referring to routes statically configured by GateD, this statement refers to routes that are learned from the FIB. Of course, it is not normally expected that routes will be learned from the FIB; however, there are two cases where this is possible. First, the routes could be remnant routes learned via the route socket after a software crash. Second, another administrative authority may have directly added routes to the FIB. This is most commonly the case when GateD is running on a UNIX system (in which case the FIB is the kernel RIB), and the super user adds routes via the route command.

In the **restrict** version of this aggregate source statement, no kernel routes will be included in the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate for which this **proto kernel** is an aggregate source. Any number of **proto kernel** aggregate sources can be used within the context of a single aggregate command.

Defaults

By default, no **proto kernel** aggregate source exists.

Context

aggregate statement

generate statement

Examples

The following example aggregates any more specific 223/8 routes learned from the kernel with the exception of 223.1/16.

```
aggregate 223/8 {  
    proto kernel {  
        all ;  
        223.1/16 exact restrict ;  
    } ;  
}
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto ospf

Name

proto ospf - filter on routes learned via OSPF

Syntax

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

Parameters

tagvalue - an arbitrary number from 0 to 4,294,967,295

preference *preference* - specifies the aggregation preference for routes that match the corresponding **proto ospf** or *route_filter*

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

Description

The **proto ospf** aggregate source statement is used to match routes that have been propagated through OSPF for inclusion in the associated aggregate route. It deals only with routes learned directly from OSPF and does not consider routes that are AS external to OSPF. For these routes, see the **proto ospfase** aggregate source statement.

In the **restrict** version of this aggregate source statement, no OSPF routes that match the optional *tag* (or no OSPF routes at all in the absence of the tag parameter) will be included in the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters that also match the optional tag parameter will be included in the aggregate route for which this **proto ospf** is an aggregate source. Any number of **proto ospf** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates matching routes are not to be contributors.

Defaults

By default, no **proto ospf** aggregate source exists.

Context

aggregate statement

generate statement

Examples

The following example includes all OSPF routes with tag 1234 that are more specific than 10/8 in the aggregate.

```
aggregate 10/8 {  
    proto ospf tag 1234 {  
        all ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto ospfase

Name

proto ospfase - filters on routes learned via OSPF that are AS External

Syntax

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

Parameters

tagvalue - an arbitrary number from 0 to 4,294,967,295

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto ospfase** or *route_filter*

Description

The **proto ospfase** aggregate source statement is used to match routes that have been propagated through OSPF as ASE for inclusion in the associated aggregate route. To aggregate routes learned as OSPF routes, see **proto ospf**.

In the **restrict** version of this aggregate source statement, no OSPF ASE routes that match the optional *tag* (or no OSPF ASE routes at all in the absence of the tag parameter) will be included in the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters that also match the optional *tag* parameter will be included in the aggregate route for which this **proto ospf** is an aggregate source. Any number of **proto ospfase** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto ospfase** aggregate source exists.

Context

aggregate statement

generate statement

Examples

The following example includes all OSPF ASE routes with tag 1234 that are more specific than 10/8 in the aggregate.

```
aggregate 10/8 {  
    proto ospfase tag 1234 {  
        all ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto rip

Name

proto rip - filter on routes learned via RIP

Syntax

```
proto rip
    [ tag tagvalue ]
    restrict ;

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

Parameters

tagvalue - an arbitrary number from 0 to 65535

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto rip** or *route_filter*

Description

The **proto rip** aggregate source statement is used to match RIP routes for inclusion in the associated aggregate route.

If the optional **tag** parameter is used, only routes matching the *tagvalue* will be considered for aggregation.

In the **restrict** version of this aggregate source statement, any RIP routes that match the optional **tag** parameter (or all RIP routes in the absence of the **tag**) will be excluded from the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters which also match the optional tag parameter will be included in the aggregate route for which this **proto rip** is an aggregate source. Any number of **proto rip** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto rip** aggregate source exists.

Context

`aggregate` statement

`generate` statement

Examples

```
aggregate 10.0.0.0 masklen 8 {  
    proto rip {  
        10/8 ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto ripng

Name

proto ripng - filter on routes learned via RIPng

Syntax

```
proto ripng
    [ tag tagvalue ]
    restrict ;

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

Parameters

tagvalue - an arbitrary number from 0 to 65535

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See “Chapter 28 Route Filtering” on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto ripng** or *route_filter*

Description

The **proto ripng** aggregate source statement is used to match RIPng routes for inclusion in the associated aggregate route.

If the optional **tag** parameter is used, only routes matching the *tagvalue* will be considered for aggregation.

In the **restrict** version of this aggregate source statement, any RIPng routes that match the optional **tag** parameter (or all RIPng routes in the absence of the **tag**) will be excluded from the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters that also match the optional **tag** parameter will be included in the aggregate route for which this **proto ripng** is an aggregate source. Any number of **proto ripng** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates that matching routes are not to be contributors.

Defaults

By default, no **proto ripng** aggregate source exists.

Context

`aggregate` statement

`generate` statement

Examples

```
aggregate feC0:: masklen 64 {  
    proto ripng {  
        feC0:: masklen 64 ;  
    } ;  
} ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

proto static

Name

proto static - filter on statically configured routes

Syntax

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

Parameters

route_filter - a set of route filters used to select particular routes or sets of routes for inclusion in or exclusion from an aggregate route. See "Chapter 28 Route Filtering" on page 129 of the *GateD Configuration Guide* for additional information.

restrict - used to specify that contributor routes are to be excluded from an aggregate. **restrict** can be protocol or *route_filter* specific.

preference preference - specifies the aggregation preference for routes that match the corresponding **proto static** or *route_filter*

Description

The **proto static** aggregate source statement is used to match routes that have been statically configured for inclusion in the associated aggregate route.

In the **restrict** version of this aggregate source statement, no static routes will be included in the associated aggregate route.

If any *route_filters* are provided, then any routes matching those filters will be included in the aggregate route for which this **proto static** is an aggregate source. Any number of **proto static** aggregate sources can be used within the context of a single aggregate command. **restrict** specified on a *route_filter* indicates matching routes are not to be contributors.

Defaults

By default, no **proto static** aggregate source exists.

Context

aggregate statement

generate statement

Examples

```
aggregate 10/8 {
    proto static {
        10/8 ;
    }
}
```

```
    } ;  
  } ;
```

See Also

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

restrict

Name

restrict - specifies routes that are not to be considered as contributors of the specified aggregate

Syntax

restrict

Parameters

none

Description

restrict specifies routes that are not to be considered as contributors of the specified aggregate. **restrict** can be used to restrict contributors on a protocol and/or prefix basis.

Defaults

none

Context

aggregate proto statement

generate proto statement

route_filter

Examples

```
aggregate 10/8 {  
    proto static restrict;  
    proto all {  
        10/8 refines;  
        10.1.2.3 restrict;  
    };  
};
```

See Also

aggregate on page 673

"Chapter 33 Route Aggregation and Generation" on page 155 in *Configuring GateD*

generate on page 678

toribs unicast | multicast

Name

toribs - specifies that the aggregate is restricted to this RIB

Syntax

```
toribs ( unicast | multicast | unicast multicast )
```

Parameters

none

Description

toribs unicast specifies that the aggregate is restricted to the unicast RIB. The default is all RIBs (unicast and multicast). **toribs multicast** specifies that the aggregate is restricted to the multicast RIB.

Defaults

The default is all RIBs (unicast and multicast).

Context

aggregate statement

Examples

Example 1

In this example, only "10/8 refines" routes in the multicast RIB contribute to the 10/8 multicast aggregate route.

```
aggregate 10/8 bgp toribs multicast {  
    proto bgp {  
        10/8 refines;  
    };  
};
```

Example 2

```
aggregate 10/8 bgp toribs unicast multicast {  
    proto bgp {  
        10/8 refines;  
    };  
};
```

See Also

aggregate on page 673

“Chapter 33 Route Aggregation and Generation” on page 155 in *Configuring GateD*

Chapter 30

Route Flap Damping

dampen-flap

Name

dampen-flap - sets options for weighted route damping

Syntax

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

Parameters

suppress-above - value of a route's instability, above which the route is suppressed

reuse-below - value of a route's instability, below which a suppressed route is reused

max-flap - the maximum value of a route's instability history

reach-decay - the time (in seconds) after which a reachable route's instability history decays to half its current value

unreach-decay - the time (in seconds) after which an unreachable route's instability history decays to half its current value

keep-history - the time (in seconds) for which any history of a route's instability is maintained

Description

Weighted route damping treats 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 for only 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.

Defaults

```
dampen flap {  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
    max-flap 16.0 ;  
    reach-decay 300;  
    unreach-decay 900;  
    keep-history 1600;  
};
```

Context

global statement

Examples

```
dampen flap {  
    suppress-above 4.0 ;  
    reuse-below 3.0 ;  
    max-flap 17.0 ;  
    reach-decay 200;  
    unreach-decay 800;  
    keep-history 2000;  
};
```

See Also

“Chapter 34 Route Flap Damping” on page 163 of *Configuring GateD*

`keep-history` on page 709

`max-flap` on page 710

`reach-decay` on page 711

`reuse-below` on page 712

`suppress-above` on page 713

`unreach-decay` on page 714

keep-history

Name

keep-history - the time (in seconds) for which any history of a route's instability is maintained

Syntax

```
keep-history time ;
```

Parameters

time - time in seconds

Description

keep-history is the time (in seconds) for which any history of a route's instability is maintained.

Defaults

```
keep-history 1600;
```

Context

dampen-flap statement

Examples

```
dampen flap{
    suppress-above 3.0 ;
    reuse-below 2.0 ;
    reach-decay 300;
    unreach-decay 900;
    keep-history 1600;
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

max-flap

Name

max-flap - the maximum value of a route's instability history

Syntax

```
max-flap flap-metric ;
```

Parameters

flap-metric - the state that is kept on a per-route basis. This metric increases by 1 each time a route transitions from a reachable state to an unreachable state.

Description

max-flap is the maximum value of a route's instability history. **max-flap**, which must be greater than the **suppress-above** threshold, determines the longest time that a route may be suppressed.

Defaults

```
max-flap 16.0 ;
```

Context

dampen-flap statement

Examples

```
dampen flap{  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
    max-flap 16.0 ;  
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

reach-decay

Name

reach-decay - the time (in seconds) after which a reachable route's instability history decays to half its current value

Syntax

```
reach-decay time ;
```

Parameters

time - time in seconds

Description

reach-decay is the time (in seconds) after which a reachable route's instability history decays to half its current value.

Defaults

```
reach-decay 300 ;
```

Context

dampen-flap statement

Examples

```
dampen flap{  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
    reach-decay 300;  
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

reuse-below

Name

reuse-below - value of a route's instability, below which a suppressed route is reused

Syntax

```
reuse-below flap-metric ;
```

Parameters

flap-metric - the state that is kept on a per-route basis. This metric increases by 1 each time a route transitions from a reachable state to an unreachable state.

Description

reuse-below is the value, specified in base-exponent form, of a route's instability, below which a suppressed route is reused. This parameter must be less than the **suppress-above** threshold.

Defaults

```
reuse-below 2.0 ;
```

Context

dampen-flap statement

Examples

```
dampen flap {  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

suppress-above

Name

suppress-above - value of a route's instability, above which the route is suppressed

Syntax

```
suppress-above flap-metric ;
```

Parameters

flap-metric - the state that is kept on a per-route basis. This metric increases by 1 each time a route transitions from a reachable state to an unreachable state.

Description

suppress-above is the value, specified in base-exponent form, of a route's instability, above which the route is suppressed.

Defaults

```
suppress-above 3.0 ;
```

Context

dampen-flap statement

Examples

```
dampen flap{  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

unreach-decay

Name

unreach-decay - the time (in seconds) after which an unreachable route's instability history decays to half its current value

Syntax

```
unreach-decay time ;
```

Parameters

time - time in seconds

Description

unreach-decay is the time (in seconds) after which an unreachable route's instability history decays to half its current value.

Defaults

```
unreach-decay 900;
```

Context

dampen-flap statement

Examples

```
dampen flap {  
    suppress-above 3.0 ;  
    reuse-below 2.0 ;  
    reach-decay 300;  
    unreach-decay 900;  
};
```

See Also

"Chapter 34 Route Flap Damping" on page 163 of *Configuring GateD*

dampen-flap on page 707

Chapter 31

SNMP Multiplexing (SMUX)

password

Name

password - specifies the clear text password to be used for authentication

Syntax

```
password string ;
```

Parameters

string - a string of up to 8 bytes that represents the password to be used

Description

The SMUX protocol allows a clear text password to be used for authentication of the sub-agent. This password is specified here.

Default

The default is a blank password (0 length).

Context

smux statement

Examples

```
smux on {  
    password "foo" ;  
}
```

See Also

smux on page 717

<http://www.ietf.org/rfc/rfc1227.txt>

port

Name

port - specifies the port on which to contact the master agent via TCP

Syntax

```
port smuxport ;
```

Parameters

smuxport - a port number on which to contact the master agent. This must be a 16-bit value.

Description

The master agent may be running on a different port than the well-known port of 199. This allows GateD to contact the agent on the specified port.

Defaults

```
port 199 ;
```

Context

smux statement

Examples

```
smux on {  
    port 2112;  
};
```

See Also

smux on page 717

<http://www.ietf.org/rfc/rfc1227.txt>

smux

Name

smux - enables or disables SMUX

Syntax

```
smux ( on | off ) ;
```

Parameters

none

Description

smux enables or disables SMUX.

Defaults

```
smux on {  
    port 199;  
};
```

Context

global

Examples

```
smux on {  
    port 199;  
};
```

See Also

<http://www.ietf.org/rfc/rfc1227.txt>

traceoptions

Name

traceoptions - specifies SMUX trace options

Syntax

```
traceoptions smuxtraceoptions ;
```

Parameters

In addition to the normal options, the following SMUX options are available:

send - Trace packets sent to the master agent.

receive - Trace packets received from the master agent.

packets - Trace every byte of traced packet.

Description

This option configures the trace options for the SMUX protocol. Packets may be traced on the receive side or send side. By specifying *packets*, the entire packet is dumped in the trace file.

Defaults

The default options are inherited from the global options.

Context

smux statement

Examples

```
smux yes {  
    traceoptions "/tmp/smuxlog" send packets;  
};
```

See Also

smux on page 717

<http://www.ietf.org/rfc/rfc1227.txt>