Lucent Technologies
Bell Labs Innovations

GRF® GateD Manual

1.4. Update 2

Part Number: 7820-2038-001
For software version 1.4.20 and later
September, 1999

Ordering Information

To order copies of this document, contact your Lucent Technologies representative or reseller.

Support Telephone Numbers

For a menu of support and other services, call (800) 272-3634. Or call (510) 769-6001 for an operator.

Lucent Technologies

Customer Service

Customer Service provides a variety of options for obtaining information about Lucent products and services, software upgrades, and technical assistance.

Finding information and software on the Internet

Visit the Web site at `http://www.ascend.com` for technical information, product information, and descriptions of available services.

Visit the FTP site at `ftp.ascend.com` for software upgrades, release notes, and addenda to this manual.

Obtaining technical assistance

You can obtain technical assistance by telephone, email, fax, modem, or regular mail, as well as over the Internet.

Enabling Lucent to assist you

If you need to contact Lucent for help with a problem, make sure that you have the following information when you call or that you include it in your correspondence:

- Product name and model.
- Software and hardware options.
- Software version.
- If supplied by your carrier, Service Profile Identifiers (SPIDs) associated with your line.
- Your local telephone company’s switch type and operating mode, such as AT&T 5ESS Custom or Northern Telecom National ISDN-1.
- Whether you are routing or bridging with your Lucent product.
- Type of computer you are using.
- Description of the problem.

Calling Lucent from within the United States

In the U.S., you can take advantage of Priority Technical Assistance or an Advantage service contract, or you can call to request assistance.

Priority Technical Assistance

If you need to talk to an engineer right away, call (900) 555-2763 to reach the Priority Call queue. The charge of $2.95 per minute does not begin to accrue until you are connected to an engineer. Average wait times are less than three minutes.

Other telephone numbers

For a menu of Lucent’s services, call (800) 272-363). Or call (510) 769-6001 for an operator.

Calling Lucent from outside the United States

You can contact Lucent by telephone from outside the United States at one of the following numbers:

Telephone outside the United States	(510) 769-8027
Austria/Germany/Switzerland	(+33) 492 96 5672
Benelux	(+33) 492 96 5674
France	(+33) 492 96 5673
Italy	(+33) 492 96 5676
Japan	(+81) 3 5325 7397
Middle East/Africa	(+33) 492 96 5679
Scandinavia	(+33) 492 96 5677
Spain/Portugal	(+33) 492 96 5675
UK	(+33) 492 96 5671

For the Asia Pacific Region, you can find additional support resources at `http://apac.ascend.com`

Obtaining assistance through correspondence

Lucent maintains two email addresses for technical support questions. One is for customers in the United States, and the other is for customers in Europe, the Middle East, and Asia. If you prefer to correspond by fax, BBS, or regular mail, please direct your inquiry to Lucent's U.S. offices. Following are the ways in which you can reach Customer Service:

- Email from within the U.S.—support@ascend.com
- Email from Europe, the Middle East, or Asia—EMEAsupport@ascend.com
- Fax—(510) 814-2312
- Customer Support BBS (by modem)—(510) 814-2302

Write to Lucent at the following address:

Attn: Customer Service
Lucent Technologies Inc.
1701 Harbor Bay Parkway
Alameda, CA 94502-3002

Important safety instructions

The following safety instructions apply to the GRF router models GRF-4-AC, GRF-4-DC, GRF-16-AC, and GRF-16-DC except as noted:

1 Read and follow all warning notices and instructions marked on the product or included in the manual.

2 Do not attempt to service this product yourself, as opening or removing covers and/or components may expose you to dangerous high voltage points or other risks. Refer all servicing to qualified service personnel.

3 The maximum recommended ambient temperature for all GRF router models is 104˚ Fahrenheit (40˚ Celsius). Care should be given to allow sufficient air circulation or space between units when the GRF chassis is installed in a closed or multi-unit rack assembly because the operating ambient temperature of the rack environment might be greater than room ambient.

4 Slots and openings in the GRF cabinet are provided for ventilation. To ensure reliable operation of the product and to protect it from overheating, maintain a minimum of 4 inches clearance on the top and sides of the GRF 400 router, and a minimum of 6 inches on the top and sides of the GRF 1600 router.

5 Installation of the GRF 400 or 1600 in a rack without sufficient air flow can be unsafe.

6 If a GRF router is installed in a rack, the rack should safely support the combined weight of all equipment it supports.
- A fully loaded, redundant-power GRF 400 weighs 38.5 lbs (17.3 kg).
- A fully loaded, single-power GRF 400 weighs 32.5 lbs (14.6 kg).
- A four card, redundant-power GRF 1600 weighs 147 lbs (66.2 kg).
- A four card, single-power GRF 1600 weighs 127 lbs (57.2 kg).

7 The connections and equipment that supply power to GRF routers should be capable of operating safely with the maximum power requirements of the particular GRF model. In the event of a power overload, the supply circuits and supply wiring should not become hazardous.

8 Models with AC power inputs are intended to be used with a three-wire grounding type plug - a plug which has a grounding pin. This is a safety feature. Equipment grounding is vital to ensure safe operation. Do not defeat the purpose of the grounding type plug by modifying the plug or using an adapter.

9 Prior to installation, use an outlet tester or a voltmeter to check the AC receptacle for the presence of earth ground. If the receptacle is not properly grounded, the installation must not continue until a qualified electrician has corrected the problem. Similarly, in the case of DC input power, check the DC ground (s).

10 If a three-wire grounding type power source is not available, consult a qualified electrician to determine another method of grounding the equipment.

11 Models with DC power inputs must be connected to an earth ground through the terminal block Earth/Chassis Ground connectors. This is a safety feature. Equipment grounding is vital to ensure safe operation.

12 Install DC-equipped GRF 400 and 1600 routers only in restricted access areas in accordance with Articles 110-16, 110-17, and 110-18 of the National Electrical Code, ANSI/NFPA 70.

13 Do not allow anything to rest on the power cord and do not locate the product where persons will walk on the power cord.

14 Industry-standard cables are provided with this product. Special cables that may be required by the regulatory inspection authority for the installation site are the responsibility of the customer.

15 When installed in the final configuration, the product must comply with the applicable Safety Standards and regulatory requirements of the country in which it is installed. If necessary, consult with the appropriate regulatory agencies and inspection authorities to ensure compliance.

Wichtige Sicherheitshinweise

Die folgenden Sicherheitshinweise gelten für die GRF-Oberfräsenmodelle GRF-4AC, GRF-4-DC, GRF-16-AC und GRF-16-DC, außer wenn anderweitig angegeben:

1 Lesen und befolgen Sie alle am Produkt angebrachten und im Handbuch enthaltenen Warnhinweise und Anleitungen.

2 Versuchen Sie nicht, dieses Gerät selbst zu warten bzw. die Abdeckung zu öffnen oder Bauteile zu entfernen. Hochspannungsgefahr. Die Wartung muß durch qualifiziertes Fachpersonal ausgeführt werden.

3 Die empfohlene maximale Umgebungstemperatur für alle GRF-Oberfräsenmodelle liegt bei 40º C. Sorgen Sie für gute Belüftung bzw. ausreichenden Abstand zwischen einzelnen Geräten, wenn das GRF-Gehäuse in einem Einzel- oder Mehrfach-Einschubrahmen installiert werden soll, da die Betriebstemperatur in dem Einschubrahmen evtl. höher als die Raumtemperatur sein kann.

4 Schlitze und Öffnungen im GRF-Gehäuse dienen zur Belüftung. Um einen einwandfreien Betrieb des Produktes zu gewährleisten und um Überhitzung vorzubeugen, jeweils oben und an den Seiten der GRF-400-Oberfräse mindestens 10,16 cm und an der GRF-1600-Oberfräse mindesten 15,24 cm Freiraum vorsehen.

5 Bei unzureichender Belüftung ist die Installation eines GRF-400 oder 1600 in einem Einschubrahmen gefährlich.

6 Bei Installation einer GRF-Oberfräse in einem Einschubrahmen, muß dieser das Gesamtgewicht aller darin installierten Geräte sicher tragen können.

– Ein komplett bestückter Redundanzstrom-GRF-400 wiegt 17,3 kg.

– Ein komplett bestückter Einzelstrom-GRF-400 wiegt 14,9 kg.

– Ein mit vier Karten bestückter Redundanzstrom-GRF-1600 wiegt 66,2 kg.

– Ein mit vier Karten bestückter Einzelstrom-GRF-1600 wiegt 57,2 kg.

7 Die Adapter und Geräte, die die GRF-Oberfräsen mit Strom versorgen, sollten auch bei maximaler Stromanforderung des einzelnen GRF-Modells noch sicher laufen. Im Fall einer Stromüberlastung sollten die Versorgungskreise und kabel keine Gefahrenquelle darstellen.

8 Alle mit Netzeingängen versehenen Geräte müssen mit einem vorschriftsmäßigen Stecker bestückt sein. Der Stecker bietet die notwendige Erdung und darf in keiner Weise modifiziert oder mit einem Adapter verwendet werden.

9 Überprüfen Sie vor der Installation mit Hilfe eines Steckdosentestgerätes oder eines Voltmeters die Erdung der Netzsteckdose. Sollte die Steckdose nicht ordnungsgemäß geerdet sein, darf mit der Installation erst fortgefahren werden, wenn ein qualifizierter Elektriker dieses Problem behoben hat. Handelt es sich um einen Gleichstromeingang ist dieser in gleicher Weise auf ordnungsgemäße Erdung zu überprüfen.

10 Ist keine 3polige geerdete Stromquelle vorhanden, beauftragen Sie einen qualifizierten Elektriker damit, das Gerät auf andere Weise zu erden.

11 Bei Modellen mit Gleichstromeingängen muß ein Erdungsdraht entweder an der Klemmleiste oder an einer Gehäuseschraube angeschlossen werden. Hierbei handelt es sich um eine Sicherheitseinrichtung. Die Erdung des Gerätes ist eine wichtige Voraussetzung für den sicheren Betrieb.

12 Die gleichstromausgerüsteten Oberfräsenmodelle GRF-400- und GRF-1600-Oberfräse dürfen nur in Bereichen mit beschränktem Zugang, unter Berücksichtigung der anwendbaren Bestimmungen für Elektroinstallationen sowie der Standards ANSI/NFPA 70 installiert werden.

13 Keine Gegenstände auf das Netzkabel stellen. Das Kabel so verlegen, daß Personen nicht versehentlich darauf treten können.

14 Standardkabel sind im Lieferumfang des Produkts enthalten. Sonderkabel, die evtl. gemäß den örtlichen Bestimmungen für die Installation erforderlich sind, sind vom Kunden zu stellen.

15 Zur Installation in der endgültigen Konfiguration muß das Produkt den am Installationsort geltenden Sicherheitsstandards und bestimmungen entsprechen. Genauere Informationen erhalten Sie ggf. bei den zuständigen Behörden.

Contents

Customer Service iii

About this Manual xix

About 1.4 Update 2 xix
What is in the manual xix
Documentation conventions xx
What you should know xxi
Documentation set xxi
Related publications xxi

Chapter 1 Introduction to GRF Dynamic Routing 1-1

Introduction to dynamic routing and GateD 1-2
- What is dynamic routing? 1-2
- How dynamic routing is implemented on the GRF 1-3
- Routing protocols 1-3
- Link state and distance vector protocols 1-4
- Descriptions of supported protocols 1-5
  - Routing Information Protocol (RIP) 1-5
  - Open Shortest Path First (OSPF) Protocol 1-5
  - Border Gateway Protocol (BGP) 1-5
  - Other routing protocols 1-6
- Routing policy (import and export) 1-6

Introduction to BGP on the GRF 1-7
- BGP protocol features 1-8
  - Autonomous systems 1-8
  - Sessions 1-10
  - Route advertisements 1-10
  - Enforcing policy 1-11
  - Path selection 1-11

Additions to GRF BGP 1-12
- Configurable Export-Best-BGP (CEBB) 1-12
- Local_Pref 1-13
- Asymmetric Multi-Level Next Hop Resolution (AMLNHR) 1-13
- BGP confederations 1-14
- GSM - monitoring tool for GateD 1-14
- MEDs 1-14
- Communities 1-15
- Route reflection 1-15

Routing arbiter interaction 1-16
Route preference biasing 1-16
Route selection 1-16
Configuring multi-exit discriminators (MEDs) 1-17
Originating a MED 1-17
BGP group statement 1-17
BGP peer statement........ 1-17
BGP export statement 1-18
Configuring route reflection 1-19
Route reflection example 1-20
/etc/gated.conf entries for GRF A - a route reflector client........ 1-21
/etc/gated.conf entries for GRF B - a route reflector client 1-22
/etc/gated.conf entries for GRF R........ 1-22
/etc/gated.conf entries for GRF C - a non-client peer........ 1-23
/etc/gated.conf entries for GRF D - a non-client peer 1-24
Introduction to OSPF 1-25
Authentication........ 1-26
Introduction to RIP 1-27
Nexthop........ 1-28
Network mask 1-28
Authentication........ 1-28
RIP I and network masks 1-29
Protocol-precedence and preference 1-30
BGP route selection algorithm........ 1-30
Assigning protocol-precedence........ 1-31
Weighted route dampening 1-32

Chapter 2 GateD Configuration Tutorial 2-1

Introduction to Configuration A 2-2
Starting Configuration A........ 2-3
Descriptions of the configuration files........ 2-4
Description of the /etc/grifconfig.conf file 2-4
Interface name 2-4
IP address 2-4
Netmask (optional) 2-5
Broadcast / destination address (optional) 2-5
Argument field (optional) 2-5
Description of /etc/gratm.conf file........ 2-5
Service section 2-5
Traffic shaping section 2-6
Signalling section 2-6
Interfaces section 2-6
PVC section 2-7
Description of /etc/gated.conf........ 2-7
Configuration A topology 2-8
Configuring and verifying interface connections........ 2-8
Editing the /etc/grifconfig.conf file........ 2-8

Editing the /etc/gratm.conf file 2-9
Verifying logical and physical connections 2-10
Configuring and verifying GateD 2-13
Editing the /etc/gated.conf file 2-13
Testing and saving the /etc/gated.conf file 2-15
Using GSM to verify /etc/gated.conf 2-16
Verifying the peering session 2-16
Determining BGP status 2-16
Summary for Configuration A 2-20

Chapter 3 GateD Configuration Statements 3-1

Syntax of GateD configuration file 3-2
Syntax description conventions 3-2
Order of statements 3-2
Statement summary 3-3
Protocol-precedence and preference 3-4
BGP route selection algorithm 3-4
Assigning protocol-precedence 3-5
Trace statement 3-6
Traceoptions syntax 3-6
Traceoptions options 3-6
Protocol significant trace options 3-7
Packet tracing 3-7
Directive statement 3-8
Options statement 3-8
Syntax 3-8
Options 3-9
GSM statement 3-9
Syntax 3-10
Options 3-10
Interface statement 3-12
Syntax 3-12
Options 3-12
Interface lists 3-14
IP interface addresses and routes 3-15
Definition statement 3-17
Syntax 3-17
Options 3-17
Example 3-18
RIP statement 3-19
Syntax 3-19
Options 3-19
Parameters 3-20
Tracing options 3-21
OSPF statement 3-22
Syntax 3-22
Options 3-23

Tracing options ... 3-27
BGP statement ... 3-28
Syntax ... 3-28
Options ... 3-30
Group and peer parameters ... 3-31
Group types ... 3-32
Specifying group parameters ... 3-33
Peer parameters ... 3-35
Specifying peer parameters ... 3-36
Tracing options ... 3-39
Recovering interfaces ... 3-39
Weighted route dampening statement ... 3-40
Syntax ... 3-40
Options ... 3-40
ICMP statement ... 3-41
Syntax ... 3-41
Option ... 3-41
Tracing options ... 3-41
Router discovery protocol ... 3-42
Router discovery server ... 3-42
Router discovery client ... 3-43
Router discovery server statement ... 3-43
Syntax ... 3-43
Options ... 3-44
Router discovery client statement ... 3-45
Syntax ... 3-45
Options ... 3-45
Tracing options ... 3-46
Kernel statement ... 3-47
Forwarding tables and routing tables ... 3-47
Syntax ... 3-47
Options ... 3-48
Tracing options ... 3-49
Static statement ... 3-51
Syntax ... 3-51
Options ... 3-51
Control statements ... 3-53
Route filtering ... 3-53
Syntax ... 3-53
Options ... 3-54
Global filters ... 3-55
Defining global filters and list ... 3-55
Matching AS paths ... 3-57
Syntax ... 3-57
AS path regular expressions ... 3-57
AS path terms ... 3-57
AS path operators ... 3-58
AS path attributes for communities ... 3-61

Syntax for import 3-62
Syntax for export........ 3-62
Options........ 3-63
Import statement 3-64
Syntax 3-64
Controlling installation of routes........ 3-64
Route filters 3-65
Importing routes from BGP syntax........ 3-65
Importing routes from BGP options 3-66
Importing routes from RIP/redirect syntax 3-66
Importing routes from OSPF syntax 3-67
Export statement 3-68
Syntax 3-68
Controlling exportation of routes syntax 3-68
Controlling exportation of routes options........ 3-68
Route filters 3-69
Specifying the destination 3-69
Exporting to BGP syntax 3-70
Exporting to BGP options........ 3-70
Exporting to RIP syntax 3-71
Exporting to OSPF syntax 3-71
Exporting to OSPF options 3-72
Specifying the source 3-72
Exporting BGP routes syntax 3-72
Exporting RIP routes 3-72
Exporting OSPF routes........ 3-73
Exporting routes from non-routing protocols........ 3-73
Non-routing by interface........ 3-73
Non-routing by protocol 3-74
Exporting by AS path 3-74
Exporting by route tag 3-74
Route aggregation and generation statements 3-75
Aggregation and generation syntax 3-76
Aggregation and generation options........ 3-77
Route filters 3-78

Chapter 4 GateD State Monitor (GSM) 4-1

Establishing a GSM connection........ 4-2
GSM connection options........ 4-2
CLI interface........ 4-2
Telnet connection........ 4-3
Using the GSM help command........ 4-5
Second-level show subcommands 4-5
Third-level commands 4-6
Displaying route tables........ 4-7
Frequently-used commands 4-8
Abbreviating commands........ 4-8

Chapter 5 Glossary of Terms 5-1

Appendix A Dynamic Routing RFCs A-1

Appendix B GateD Configuration File B-1

Syntax description conventions B-2
Order of statements B-2
# Trace options statement B-3
# Options statement B-3
# GSM statement B-3
# Interface statement B-3
# Definition statement B-4
# RIP statement B-4
# OSPF statement B-5
# BGP statement B-6
# Weighted route dampening statement B-8
# ICMP statement B-8
# Router discovery server statement B-9
# Router discovery client statement B-9
# Kernel statement B-9
# Static statement B-10
# Filtering statement B-10
# Autonomous system matching statement B-10
# AS path attribute statement for importing routes B-10
# AS path attribute statements for exporting routes B-11
# Importing from BGP statement B-12
# Importing from RIP and Redirect statement B-12
# Importing from OSPF statement B-12
# Export statement B-13
# Exporting to BGP statement B-13
# Exporting to RIP statement B-13
# Exporting to OSPF statement B-13
# Exporting BGP routes B-14
# Exporting RIP routes B-14
# Exporting OSPF routes B-14
# Exporting routes, non-routing by interface B-14
# Exporting routes, non-routing by protocol B-15
# Exporting by AS path B-15
# Exporting by route tag B-15
# Aggregation and generation statements B-15
# Directive statement B-16
# Include statement B-16

Appendix C Configuration files for A C-1

/etc/grifconfig.conf file C-1
/etc/gratm.conf file C-4
/etc/gated.conf file C-11
GSM displays for configuration A C-12

Index Index-1

Figures

Figure 1-1 Simple network configuration 1-2
Figure 1-2 IGPs and EGPs on a network.... 1-4
Figure 1-3 Supported protocols on a network 1-6
Figure 1-4 Full mesh and reflected route topologies.... 1-19
Figure 1-5 Client and non-client reflected route topologies.... 1-20
Figure 1-6 Route reflection configuration example 1-21

Figure 2-1 Planned configurations for GateD tutorial.... 2-2
Figure 2-2 Topology of Configuration A 2-8

Tables

Table 1-1 Precedence values 1-31

Table 3-1 Summary of GateD configuration statements 3-3
Table 3-2 Precedence values 3-5

About this Manual

The *GRF GateD Manual* provides an overview of GateD as implemented on the GRF, a configuration tutorial, and descriptions of the configuration statements.

About 1.4 Update 2

The 1.4 GRF manual set is updated to include new features added since software release 1.4.12.

This manual describes the full set of GateD features for GRF® units running software version 1.4.20 and later. User documentation for any feature added in a subsequent release will appear in an addendum. Some features might not be available with older versions or with specialty loads of the software.

Unless otherwise noted, the information in this guide applies to GRF 400 and GRF 1600 systems, as well as GRF 400 and GR-II systems using an RMS node.

What is in the manual

The *GRF GateD Manual* contains the following chapters and appendices:

- Chapter 1, “Introduction to GRF Dynamic Routing,” provides an overview of how GateD is implemented on the GRF system, and descriptions of the supported dynamic routing protocols (BGP, OSPF, and RIP).
- Chapter 2, “GateD Configuration Tutorial,” provides information and examples of how to configure various networks using GRF GateD functionality and features.
- Chapter 3, “GateD Configuration Statements,” defines and explains all the configuration statements used in the `/etc/gated.conf` configuration file.
- Chapter 4, “GateD State Monitor (GSM),” describes the GSM interface used to query internal GateD variables, and provides examples of some of the GSM outputs.
- Chapter 5 contains a glossary of related GRF and GateD terms.
- Appendix A lists the dynamic routing RFCs.
- Appendix B is an `/etc/gated.conf` configuration file that contains all the statements and parameters.
- Appendix C includes the configuration files associated with the tutorial in chapter 2.

Documentation conventions

This section explains all the special characters and typographical conventions in this manual.

Convention	**Meaning**
`Monospace text`	Used for all messages from the computer, program code, menu options, fields on screens, system prompts, file names, directory names, and paths. Variables within computer output are shown in *`italic`* of the same font.
`Boldface mono-space text`	The **`user input font`** is used for all commands and entries that a user must type. Variables within commands and entries that a user must type are shown in *`non-bold italic`*.
[ ]	Square brackets ([ ]) indicate optional elements.
|	Vertical bars (|) separate options. You can only choose one of the options; the alternatives are mutually exclusive elements. There are two kinds of choices used: — Choices that are a mandatory part of the syntax are presented in curly brackets ({ }). A choice must be made from the options presented in the curly brackets. — Choices that are an optional part of the syntax are presented in square brackets ([ ]). The portion of the syntax within square brackets is optional. If it is used, a choice must be made from the options presented in the square brackets.
Key1-Key2	Represents a combination keystroke. To enter a combination keystroke, press the first key and hold it down while you press one or more other keys. Release all the keys at the same time. (For example, Ctrl-H means hold down the Control key and press the H key.)
Note:	Introduces important additional information.
⚠ **Caution:**	Warns that a failure to follow the recommended procedure could result in loss of data or damage to equipment.
⚡ **Warning:**	Warns that a failure to take appropriate safety precautions could result in physical injury.

What you should know

This guide is for the administrator who configures and maintains dynamic routing on GRF systems.

To use and configure GRF GateD, you need to understand the following:

- IP internetworking.
- IP protocols and routing operations.
- A basic understanding of dynamic routing and the RIP, OSPF, and BGP protocols.

In addition, configuring and monitoring the GRF requires that a Network Administrator have experience with and an understanding of UNIX systems, and the ability to navigate in a UNIX environment. Knowledge of UNIX, its tools, utilities, and editors is useful, as is experience with administering and maintaining a UNIX system.

The Network Administrator must understand how TCP/IP internetworks are assembled; what interconnections represent legal topologies; how networks, hosts, and routers are assigned IP addresses and configured into operation; and how to determine and specify route table (routing) information about the constructed internetwork(s). Although not required, a high-level understanding of SNMP is useful.

Documentation set

The GRF 1.4 Update 2 documentation set consists of the following manuals:

- *GRF 400/1600 Getting Started - 1.4 Update 2*
- *GRF Configuration and Management Guide - 1.4 Update 2*
- *GRF Reference Guide - 1.4 Update 2*
- *GRF GateD Manual - 1.4 Update 2* (this manual)

Related publications

The following are related publications that you may find useful:

- *Internet Routing Architectures*, by Bassam Halabi, Cisco Press, 1997. Recommended for BGP information.
- *Routing in the Internet*, by Christian Huitema. Prentice Hall PTR, 1995. Recommended for information about IP, OSPF, CIDR, IP multicast, and mobile IP.
- *OSPF: Anatomy of an Internet Routing Protocol*, by John T. Moy, Addison Wesley Longman, Inc., 1998.
- *Internetworking with TCP/IP*, Volume 1 and 2, by Douglas E. Comer, and David L. Stevens. Prentice-Hall,

- *TCP/IP Illustrated,* Volumes 1 and 2, by W. Richard Stevens. Addison-Wesley, 1994.
- *Interconnections*, Radia Perlman. Addison-Wesley, 1992. Recommended for information about routers and bridging.
- *TCP/IP Network Administration*, by Craig Hunt. O'Reilly & Associates, Inc. 1994. Recommended for network management information.

Introduction to GRF Dynamic Routing

1

Chapter 1 describes the following topics:

Introduction to dynamic routing and GateD 1-2

Introduction to BGP on the GRF 1-7

Additions to GRF BGP 1-12

Configuring multi-exit discriminators (MEDs) 1-17

Configuring route reflection 1-19

Introduction to OSPF 1-25

Introduction to RIP. 1-27

Protocol-precedence and preference 1-30

Weighted route dampening 1-32

Introduction to dynamic routing and GateD

The purpose of this section is to introduce the concept of dynamic routing, to discuss why it is important to network systems, and introduce some of the basic dynamic routing architectural features, including the protocols supported by the GRF system. To accomplish this, the following topics are described:

- What is dynamic routing and why is it important?
- A description of how dynamic routing is implemented on the GRF system.
- A definition of interior gateway protocols (IGPs) and exterior gateway protocols (EGPs), and the differences between them.
- A definition of link state and distance vector protocols, and the differences between them.
- A brief description of the protocols supported by GateD.
- A brief description of routing policy (import and export).

What is dynamic routing?

Dynamic routing is a mechanism that automatically adjusts to changes in a network topology so that traffic on the network continues to flow without interruption. Without dynamic routing, a network administrator would have to manually update all the necessary routing information by hand if, for example, a new machine is added to a networked system, or part of the network crashes and traffic must be rerouted.

A simple example of a network is shown in Figure 1-1. This network has been configured so that traffic from workstation 1 to workstation 2 goes through routers A, B, and D (over the darker dashed lines in Figure 1-1). If router B crashes, traffic between workstation 1 and 2 is interrupted until routers A and D are reconfigured to pass traffic through router C instead (over the lighter dashed lines in Figure 1-1).

An administrator can do this manually, but to minimize downtime, it is more efficient for dynamic routing to automatically reroute the traffic. Thus, the flow of traffic is uninterrupted, and the end user never realizes that part of the network crashed. In other words, the customer is provided continuous service.

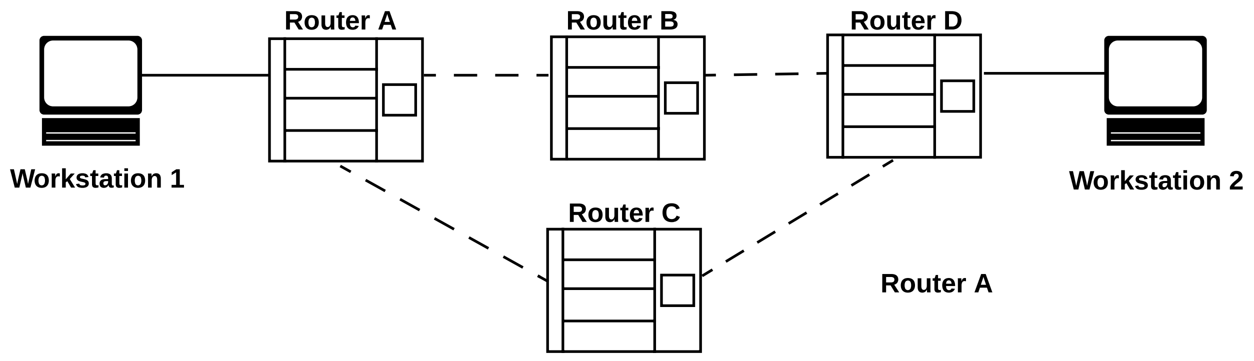

Figure 1-1. Simple network configuration

For dynamic routing to accomplish its tasks, it employs an agent. A dynamic routing agent is software that uses dynamic routing protocols to exchange information about the state of the network. It then processes that information to decide how to configure a given router to route traffic through the network efficiently. The following sections describe in more detail how dynamic routing agents work and how dynamic routing is implemented on a GRF system.

How dynamic routing is implemented on the GRF

Dynamic routing is implemented on a network system through a dynamic routing agent. On the GRF system, the agent is called the Gate Daemon or simply GateD, and is based on the software produced by the Merit GateDaemon Project.

GateD is designed to handle dynamic routing with a routing database built from information exchanged by routing protocols. GateD supports the use of the following dynamic routing protocols:

- Border Gateway Protocol (BGP)
- Routing Information Protocol (RIP)
- Open Shortest Path First (OSPF) protocol

See “Description of Supported Protocols” on page 1-5 for more information on these protocols.

GateD is a modular software program consisting of core services, a routing database, and protocol modules supporting multiple routing protocols (which are defined in the previous list). GateD allows the network administrator to configure routing policy on the GRF through import/export statements that control learning and advertising (or redistributing) of routing information by individual protocol, by source and destination autonomous system (AS), source and destination interface, previous hop router, and specific destination address. For a definition of an autonomous system, see the following section, “Routing Protocols.”

Lucent has added many enhancements and features to portions of GateD, and particularly to the BGP routing protocol (which is discussed in the “Description of Supported Protocols”on page 1-5.

Routing protocols

Routing protocols determine the best route to each destination and distribute routing information among the systems on a network. Routing protocols are divided into two general groups: interior gateway protocols (IGPs) and exterior gateway protocols (EGPs). GateD combines the management of all protocols. The following paragraphs briefly describe the protocols, and the differences between them.

IGPs are used to exchange routing information within an autonomous system (AS). An AS is a collection of networks under a common administration sharing a common routing strategy; see “Autonomous systems,” on page 1-8 for more information. The GRF’s implementation of GateD supports the Routing Information Protocol (RIP) versions 1 and 2, and Open Shortest Path First (OSPF) IGPs. All network components within an AS can use one or more IGPs to route traffic flow.

EGPs are used to exchange routing information between ASs. EGPs are only required when an AS must exchange routing information with another AS. EGPs enable packets headed for destinations only reachable by “foreign” networks to be passed to the proper router in the foreign network.

Figure 1-2 shows the relationship between an AS and, IGPs and EGPs.

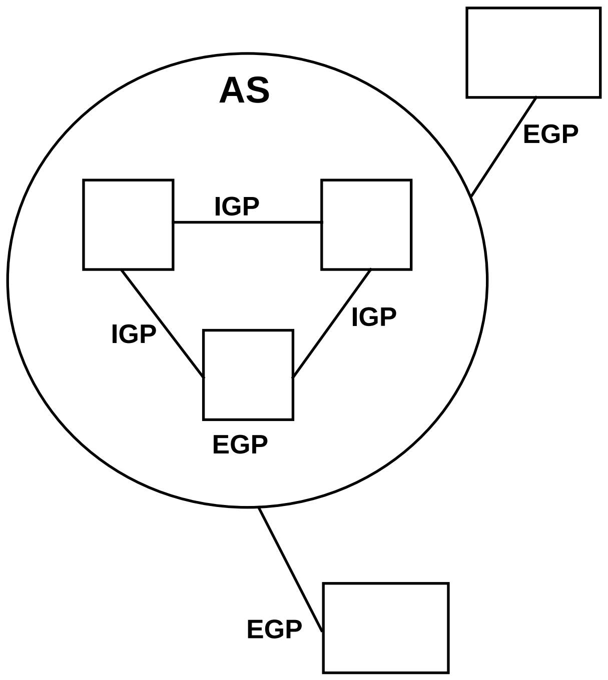

Figure 1-2. IGPs and EGPs on a network

The GRF's implementation of GateD supports the following EGPs: Exterior Gateways Protocol (EGP) and Border Gateway Protocol (BGP). EGP is an older exterior gateway protocol and BGP has replaced EGP as the exterior protocol of choice for most vendors.

Link state and distance vector protocols

Another way to categorize routing protocols is by the way route table information is managed and the kind of information that is exchanged. The two main types are link state and distance vector.

Link state protocols exchange information about the status of the links in the network. This information is propagated to all routers in the network so that each knows the topology of the entire network. From this information, they can compute the shortest path across the network for each destination and route packets along those paths. By maintaining more complete information, link state protocols can make better routing decisions but must maintain information that may be difficult to manage in large, dynamic networks. The OSPF routing protocol is an example of a link state protocol.

In distance vector protocols each router advertises the shortest path it knows to each destination. As routes are advertised and learned, the router selects the routes with the shortest distance to each destination. The router is not burdened with tracking the state of all of the links in the network, but is making routing decisions with less complete information.

The RIP and BGP routing protocols are examples of distance vector protocols. BGP is a more sophisticated "path vector protocol" which exchanges path information and not just distances. This lets it address some of the limitations of distance vector protocols.

Each protocol has advantages and disadvantages. The choice of the appropriate protocol will depend on many factors such as the size, complexity, and stability of the network.

Descriptions of supported protocols

The following sections describe the four major routing protocols that are supported by GateD.

Routing Information Protocol (RIP)

RIP versions 1 and 2 are the most commonly used distance vector interior routing protocol. RIP selects the route with the lowest number of router hops between the current router and the destination as the “best route”.

RIP assumes the best route is the one that uses the fewest hops. It does not make use of information such as line speeds or congestion conditions, and therefore is usually used in simple networks.

For more information on RIP, see “Introduction to Routing Information Protocol (RIP)” on page 1-27.

Open Shortest Path First (OSPF) Protocol

OSPF is a link state interior routing protocol. The state and condition of links between routes are used to determine the “best route”. OSPF is better suited than RIP for routing information in a complex network with many routers in a single AS. OSPF chooses the least cost path as the best path. It provides equal cost multipath routing where packets to a single destination can be sent through more than one interface simultaneously.

For more information on OSPF, see “Introduction to OSPF” on page 1-25.

Border Gateway Protocol (BGP)

BGP is a distance vector exterior routing protocol used for exchanging routing information between autonomous systems (ASs). It provides more capabilities than the Exterior Gateway Protocol (EGP) and has replaced EGP as the exterior protocol of choice for most vendors.

BGP uses path attributes to provide more information about each route as an aid in selecting the best route. Path attributes are administrative preferences based on political, organizational, or security considerations in the routing decisions. BGP supports non hierarchical network topologies and can be used to implement complex network structures, including ASs. The GRF’s version of BGP has been enhanced to include many more configuration options than the standard version of BGP.

In some situations, it may be useful to use BGP as an IGP to route traffic within an AS. It is then referred to as Internal BGP (IBGP). See “Introduction to GRF BGP” on page 1-7 for more information on BGP.

Figure 1-3 shows how the various protocols can be used within a network.

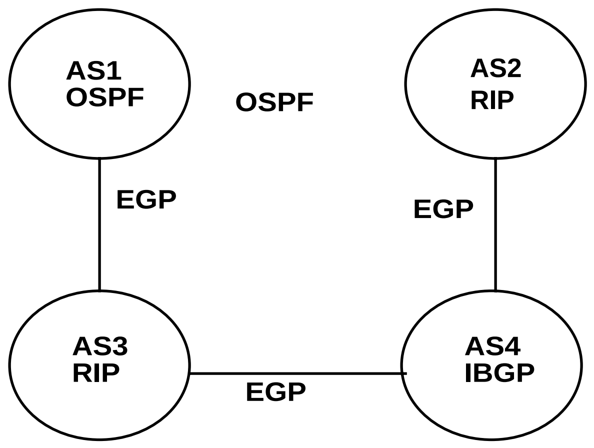

Figure 1-3. Supported protocols on a network

Other routing protocols

The Router Discovery protocol is used to inform hosts of the availability of hosts it can send packets to and to supplement a statically-configured default router. This is the preferred protocol for hosts to run. Router Discovery is described in RFC 1256. See Chapter 3 for more information on this protocol.

Routing policy (import and export)

Dynamic routing agents allow network administrators to control how they learn routes from, and advertise (or redistribute) routes to other dynamic routing agents. This is known as routing policy; it is also referred to as the import/export policy. Import statements allow the user to control the way routes are learned, while export statements allow the user a way to control the way to advertise (or redistribute) routes. By configuring the routing policy on a network’s routing agents, network administrators control the flow of routing information in the network based on characteristics such as the route prefix being advertised, the source and/or destination of the advertisement, the protocol by which the route was learned, or the source/destination autonomous system (AS).

Introduction to BGP on the GRF

The Border Gateway Protocol (BGP) is an exterior routing protocol used for exchanging routing information between autonomous systems. BGP is used for exchange of routing information between multiple transit autonomous systems as well as between transit and stub autonomous systems. BGP is related to EGP, but has more capability, greater flexibility, and less required bandwidth. BGP uses path attributes to provide more information about each route, and in particular to maintain an AS path, that includes the AS number of each autonomous system the route has transited, providing information sufficient to prevent routing loops in an arbitrary topology. Path attributes can also be used to distinguish between groups of routes to determine administrative preferences, allowing greater flexibility in determining route preference to achieve a variety of administrative ends.

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

The BGP implementation supports three versions of the BGP protocol, versions 2, 3 and 4. BGP versions 2 and 3 are quite similar in capability and function. They only propagate classed network routes, and the AS path is a simple array of AS numbers. BGP 4 propagates fully general address-and-mask routes, and the AS path structure can represent the results of aggregating dissimilar routes.

External BGP sessions may or may not include a single metric, that BGP calls the multi-exit discriminator (MED), among the path attributes. For BGP versions 2 and 3, this metric is a 16-bit unsigned integer. For BGP version 4, it is a 32-bit unsigned integer. Smaller values of the MED are preferred. Currently this metric is only used to break ties between routes with equal preference from the same neighboring AS.

Internal BGP sessions carry at least one metric in the path attributes, that BGP calls the LocalPref. The range of LocalPref is identical to the range of the MED. For BGP versions 2 and 3, a route is preferred if its value for LocalPref is smaller. For BGP version 4, a route is preferred if its value for this metric is larger. BGP version 4 internal sessions can optionally include a second metric, the MED, carried in from external sessions. The use of these metrics is dependent on the type of internal protocol processing that is specified.

BGP collapses routes with similar path attributes into a single update for advertisement. Routes that are received in a single update are readvertised in a single update. The churn caused by the loss of a neighbor is minimized and the initial advertisement sent during peer establishment are maximally compressed. BGP does not read information from the kernel message-by-message, but fills the input buffer. It processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all incoming data queued on the socket. This feature can cause other protocols to be blocked for prolonged intervals by a busy peer connection.

All unreachable messages are collected into a single message and sent prior to reachable routes during a flash update. For these unreachable announcements, the nexthop is set to the local address on the connection, no metric is sent, and the path origin is set to incomplete. On external connections the AS path in unreachable announcements is set to the local AS. On internal connections, the AS path is set to length zero.

The BGP implementation expects external peers to be directly attached to a shared subnet, and expects those peers to advertise nexthops that are host addresses on that subnet (though this constraint can be relaxed by configuration for testing). For groups of internal peers, however, there are several alternatives that can be selected from by specifying the group type and route reflection options. Type `internal` groups expect all peers to be directly attached to a shared subnet so that, like external peers, the nexthops received in BGP advertisements can be used directly for forwarding. Type `routing` groups instead determine the immediate nexthops for routes by using the nexthop received with a route from a peer as a forwarding address, and by using this to look up an immediate nexthop in an IGP's routes. Such groups support distant peers, but need to be informed of the IGP or IGPs whose routes they are using to determine immediate nexthops.

For internal BGP group types (and for test groups), where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group, with possible adjustments to the next-hop field as appropriate to each peer. This minimizes the computational load of running large numbers of peers in these types of groups. BGP allows unconfigured peers to connect if an appropriate group has been configured with an allow clause.

BGP protocol features

The Border Gateway Protocol (BGP) is an exterior routing protocol used for exchanging routing information between autonomous systems. BGP exchanges routing information between multiple transit autonomous systems as well as between transit and stub autonomous systems. While BGP is related to older EGP protocol, it operates with more capability, greater flexibility, and less required bandwidth. BGP uses path attributes to provide more information about each route and, in particular, maintains an AS path. The AS path includes the AS number of each autonomous system the route transits, providing information sufficient to prevent routing loops in an arbitrary topology.

Path attributes establish administrative preferences among groups of routes, and allow flexibility in assigning route preference to achieve a variety of administrative ends.

The BGP protocol provides a high degree of control and flexibility for doing interdomain routing while enforcing policy and performance constraints and avoiding routing loops.

The following sections describe BGP features.

Autonomous systems

The classic definition of an autonomous system (AS) is a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASs.

It is common for a single AS to use several interior gateway protocols and sometimes several sets of metrics within the AS. The term, autonomous system, stresses that even when multiple IGPs and metrics are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and presents a consistent picture of which networks are reachable through the AS.

A connection between two autonomous systems has the following two aspects:

– A physical connection

  There is a shared network between the two ASs and, on this shared network, each AS has at least one border gateway belonging to that AS. In this way, the border gateway of each AS can forward packets to the border gateway of the other AS without resorting to Inter-AS or Intra-AS routing.

– A BGP connection

  There is a BGP session between BGP speakers in each of the ASs, and this session communicates those routes that can be used for specific networks through the advertising AS. The BGP speakers that form the BGP connection must themselves share the same network that their border gateways share. Thus, a BGP session between adjacent ASs requires no support from either inter-AS or intra-AS routing.

At each connection, each AS has one or more BGP speakers and one or more border gateways, and these BGP speakers and border gateways are all located on a shared network. Paths announced by a BGP speaker of one AS on a given connection are taken to be reachable for each of the border gateways of the other ASs on the same shared network. Indirect neighbors are allowed.

Much of the traffic carried within an AS either originates or terminates at that AS. Either the source or destination IP address of the IP packet identifies a host on a network internal to that AS. Traffic that fits this description is called local traffic. Traffic that does not fit this description is called transit traffic. One goal of BGP usage is to control the flow of transit traffic.

Based on the way a particular AS deals with transit traffic, the AS fits one of the following categories:

– stub AS
  A stub AS has a single connection to one other AS and only carries local traffic.

– multihomed AS
  A multihomed AS has connections to more than one other AS, but refuses to carry transit traffic.

– transit AS
  A transit AS has connections to more than one other AS, and is designed (under certain policy restrictions) to carry both transit and local traffic.

Since a full AS path provides an efficient and straightforward way of suppressing routing loops and eliminates the count-to-infinity problem associated with some distance vector algorithms, BGP imposes no topological restrictions on the interconnection of ASs.

Sessions

BGP supports two basic types of sessions between neighbors, internal (also known as IBGP) and external (also known as EBGP). Internal sessions are run between routers in the same autonomous system; external sessions run between routers in different autonomous systems. When sending routes to an external peer, the local AS number is prepended to the AS path, thus routes received from an external peer are guaranteed to have the AS number of that peer at the start of the path.

Routes received from an internal neighbor do not generally have the local AS number prepended to the AS path. These routes usually have the same AS path that the route had when the originating internal neighbor received the route from an external peer.

Routes with no AS numbers in the path may be legitimately received from internal neighbors; these indicate that the received route should be considered internal to your own AS.

Route advertisements

BGP collapses routes with similar path attributes into a single update for advertisement. Routes that are received in a single update are readvertised in a single update. The churn caused by the loss of a neighbor is minimized, and the initial advertisement sent during peer establishment is maximally compressed. BGP does not read information from the kernel message-by-message, but fills the input buffer and processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all incoming data queued on the socket. This feature can cause other protocols to be blocked for prolonged intervals by a busy peer connection.

The BGP implementation expects external peers to be directly attached to a shared subnet, and expects those peers to advertise next hops that are host addresses on that subnet (this constraint can be relaxed by configuration for testing).

For groups of internal peers, alternatives can be selected by specifying the group type:

– Type internal groups expect all peers to be directly attached to a shared subnet so that, like external peers, the next hops received in BGP advertisements can be used directly for forwarding.

– Type routing groups determine the immediate next hops for routes by using the next hop received with a route from a peer as a forwarding address, and use this to look up an immediate next hop in an internal gateway’s routes. These groups support distant peers, and need to be informed of the IGP whose routes they are using to determine immediate next hops.

For internal BGP group types, where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group, with possible adjustments to the next hop field as appropriate to each peer. This message minimizes the computational load of running large numbers of peers in these types of groups. BGP allows unconfigured peers to connect if an appropriate group is configured with an allow clause.

Enforcing policy

BGP provides the capability for enforcing policies based on various routing preferences and constraints. Policies are not directly encoded in the protocol. Rather, policies are provided to BGP in the form of configuration information. BGP enforces policies by affecting the selection of paths from multiple alternatives and by controlling the redistribution of routing information.

Policies are determined by the AS administrator. Routing policies are created according to political, security, or economic considerations. The following are examples of routing policies that can be enforced with the use of BGP:

1 A multihomed AS can refuse to act as a transit AS for other ASs. It does so by only advertising routes to networks internal to the AS.

2 A multihomed AS can become a transit AS for a restricted set of adjacent ASs. It does so by advertising its routing information only to this set of ASs.

3 An AS can favor or disfavor the use of certain ASs for carrying transit traffic from itself.

Performance-related criteria can also be controlled using BGP. For example, an AS can minimize the number of transit ASs by preferring shorter AS paths over longer paths.

Path selection

A BGP speaker's major task is to evaluate different paths to a destination network from its border gateways at that network, select the best path, apply appropriate policy constraints, and then advertise that path to all its BGP neighbors.

The key issue is how different paths are evaluated and compared. In traditional distance vector protocols (RIP), only one metric (hop count) is associated with a path. As such, comparison of different paths is reduced to simply comparing two numbers. A complication in inter-AS routing arises from the lack of a universally agreed upon metric among ASs that can be used to evaluate external paths. Therefore, each AS can have its own set of criteria for path evaluation.

A BGP speaker builds a routing database consisting of the set of all feasible paths and the list of networks reachable through each AS sequence.

Additions to GRF BGP

The GRF implementation of BGP includes the following additional functions:

- Configurable export-best BGP (CEBB): send best BGP route when route from another protocol is active
- Asynchronous multilevel nexthop resolution: nexthopself within IBGP
- AS path truncate variable
- BGP enhancements: policy per peer, peer groups with separate policy per peer, send original BGP nexthop
- AS path length algorithm
- Increased number of GateD adjacencies/peering sessions
- GateD State Monitor (GSM) tool
- BGP confederations
- BGP multi-exit discriminator (MED)
- Communities
- Route filtering
- Route reflection
- Routing arbiter interaction
- Route preference biasing
- Stability support for BGP-OSPF interaction
- A weighted route dampening statement

Configurable Export-Best-BGP (CEBB)

The configurable export-best-BGP (CEBB) functionality allows GateD to export BGP routes that are not actually installed because there is an IGP route that takes precedence over the BGP route. Essentially, if a route prefix is known from a protocol other than BGP and is more preferred than BGP (for example, an IGP), CEBB allows the most preferred BGP route to be exported without having to redistribute the IGP into the BGP. This is especially useful for external BGP peering relationships where IGP routes are not generally redistributed.This allows for simpler policy configuration because explicit protocol redistribution is no longer required.

Additionally, CEBB increases BGP stability both internally and externally. For example, if an IGP is redistributed into BGP, and there is some level of instability within the IGP, it would appear as a BGP flap to external peers.In this case, CEBB can increase stability because the external BGP peer always receives the best BGP route regardless of whether a matching IGP route was installed.

This functionality is on by default; to disable it, you must add the **`disable export best`** clause as a global statement in the BGP portion of the `/etc/gated.conf` file.

Local_Pref

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

GateD always uses the received **`Local_Pref`** to select between BGP routes. BGP routes with a larger **`Local_Pref`** are preferred. The range of values for local preference are 0 - 2**31.

Note: All routers in the same network that are running GateD and participating in IBGP should use **`localpref`** uniformly. That is, if one router has **`localpref`** set, all should set it, and all should use the same value.

Asymmetric Multi-Level Next Hop Resolution (AMLNHR)

This enhancement to the GRF BGP route selection algorithm allows layers of nexthops to be resolved by other BGP prefixes until it is finally resolved by an IGP prefix. This is consistent with the default behavior of other common BGP implementations.

In the past, you were only able to walk up the “Next-hop-resolution-tree” one level because the GRF did not support the resolution of next hops with BGP. So, if a user attempted to resolve a prefix learned through BGP, it would only compare the resolving BGP prefix with other prefixes that could possibly resolve the next hop one level above.

Shortest IGP distance - The route whose NEXT_HOP is closer, with respect to the IGP distance, is preferred.

The proper comparison of the “ultimate” resolving IGP routes is used. The depth of the search is completely dependent on the resolution of a prefix and is not hard-coded to a fixed depth limit.

Old style:

BGP prefix NH = A -> NH resolution of A
(actual depth required for “ultimate” resolution disregarded)

New style:

BGP prefix NH = A -> NH resolution of A -> B = Resolution of A's NH -> C = Resolution of B's NH -> D = OSPF prefix

The purpose of this feature allows customers to route completely based upon closest exit or administratively-configured IGP variables. Also, it should add stability to customer networks because if a level within the resolution tree is withdrawn by a neighbor or peer, there is a greater likelihood of immediate resolution within GateD instead of potential flood storms until convergence.

BGP confederations

In addition to improvements in routing policy control, current techniques for deploying BGP among speakers in the same autonomous system generally require a full mesh of TCP connections among all speakers for the purpose of exchanging exterior routing information.

In autonomous systems, the number of intra-domain connections that need to be maintained by each border router can become significant. While this is usually acceptable in small networks, it may lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports confederations for internal peer groups (with BGP version 4 only). It may be useful to subdivide an autonomous system into smaller domains for purposes of controlling routing policy through information contained in the BGP AS_PATH attribute (similar to EBGP). For example, one may chose to consider all BGP speakers in a geographic region as a single entity (confederation).

Subdividing a large autonomous system allows a significant reduction in the total number of intra-domain BGP connections, as the connectivity requirements simplify to the model used for inter-domain connections.

There is usually no need to expose the internal topology of this divided autonomous system, which means it is possible to regard a collection of autonomous systems under a common administration as a single entity or autonomous system when viewed from outside the confines of the confederation of autonomous systems itself.

A member of a BGP confederation will use its routing domain identifier (the internally visible AS number) in all transactions with peers that are members of the same confederation as the given router.

A BGP speaker receiving an AS_PATH attribute containing a confederation ID matching its own confederation treats the path in the same fashion as if it had received a path containing its own AS number. Thus, BGP peering sessions at the confederation border are analogous to external BGP transactions within an AS; and BGP peering sessions within the confederation are analogous to internal BGP transactions.

The confederation implementation conforms to RFC 1966, “Autonomous System Confederations for BGP”.

GSM - monitoring tool for GateD

The GateD State Monitor (GSM) provides an interactive interface to a running GateD daemon which can be used to query internal GateD variables. See Chapter 4 for more information on GSM.

MEDs

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

This attribute is not propagated to other neighboring ASs. However, this attribute can be propagated to other BGP speakers within the same AS.

The MED attribute for BGP version 4 is a four-octet, unsigned integer.

MED is originated using the **`metricout`** option of the export, group and/or peer statement. It is imported using the **`med`** keyword on the BGP group statement. A MED can also be created from IGP metrics when exporting one protocol into another, for example, a redistribution of OSPF into BGP.

Communities

A community is a group of destinations that share a common property. For example, all destinations belong by default to the general Internet community. Each destination can belong to multiple communities. All prefixes with the community attribute belong to the communities listed in the attribute. An autonomous system administrator can define those communities to which a particular destination belongs.

The communities attribute allows the administrator of a Routing Domain to tag groups of routes with a community tag. The tag consists of 2 octets of autonomous system (AS) and 2 octets of community ID. The community attribute is passed from routing domain to routing domain to maintain the grouping of these routes. A set of routes can have more than one community tag in its community attribute.

Communities import policy is configured using the **`aspath-opt`** clause.

Communities export policy is configured using **`aspath-opt`** and/or **`mod-aspath`** clauses.

Please refer to the Communities specification and its accompanying usage documents (RFC 1997 and RFC 1998 as of this writing) for further details on BGP communities.

Route reflection

Generally, all border routers in a single AS need to be internal peers of each other, and in fact all non-border routers frequently need to be internal peers of all border routers. While this is usually acceptable in small networks, it can lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports route reflection for internal peer groups (with BGP version 4 only). When using route reflection, the rule that a router cannot readvertise routes from internal peers to other internal peers is relaxed for some routers, called route reflectors. A typical use of route reflection might involve a core backbone of fully meshed routers. Fully meshed means all the routers in the fully meshed group peer directly with all other routers in the group, some of which act as route reflectors for routers that are not part of the core group.

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

Typically, a single router acts as the reflector for a set, or cluster of clients. However, for redundancy two or more can also be configured to be reflectors for the same cluster. In this case, a cluster ID should be selected to identify all reflectors serving the cluster, using the

`clusterid` keyword. Gratuitous use of multiple redundant reflectors is not advised, as it can lead to an increase in the memory required to store routes on the redundant reflectors' peers.

No special configuration is required on the route reflection clients. From a client's perspective, a route reflector is simply a normal IBGP peer. Any BGP version 4 speaker should be able to be a reflector client. (Note however that GateD versions 3.5B3 and earlier, and 3.6A1 and earlier, contain a problem that prevents them from acting as route reflection clients.)

Readers are referred to the route reflection specification document (RFC 1965 as of this writing) for further details.

Routing arbiter interaction

GateD can interact with the Routing Arbiter's Route Server, as deployed at various Internet backbone Network Access Points (NAPs). GateD properly handles BGP routing updates from the Route Server. Previously, GateD would drop the Route Server peer session.

The **`ignorefirstashop`** option is added and enables GateD to retain routes propagated by the Route Server.

Route preference biasing

The administrator of a BGP routing domain can modify the length of an exported AS Path by including the AS number multiple times in the AS Path using the **`ascount`** option. This can influence the neighbor's choice of preferred route, and help prevent sub-optimal routing over redundant links between neighboring autonomous systems.

Route selection

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors. GateD uses the following criteria to select the best path. If routes are equal at a given point in the selection process, then the next criterion is applied to break the tie:

1 Configured policy: the route with smallest preference, as determined by the policy defined in `/etc/gated.conf`.

2 **`Local_Pref`**: the route with the highest BGP local preference.

3 Shortest AS Path: the route with the fewest ASs listed in its AS path.

4 Origin IGP < EGP < Incomplete: the route with an AS path origin of IGP is preferred. Next in precedence is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.

5 MED (if not ignored): the route with the best multi-exit discriminator (MED) is preferred. MEDs are only compared between routes that are received from the same neighbor AS. That is, this test is only applied if the local AS has two or more connections to a given neighbor AS.

6 Shortest IGP distance: the route whose nexthop is closer (with respect to the IGP distance) is preferred.

7 Source IGP < EBGP < IBGP: first, prefer the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.

8 Lowest Router ID: the route whose nexthop IP address is numerically lowest.

Configuring multi-exit discriminators (MEDs)

External BGP sessions may or may not include a single metric in the path attributes, which BGP calls the Multi-Exit Discriminator (MED).

For BGP versions 2 and 3, this metric is a 16-bit unsigned integer, and for BGP version 4, it is a 32-bit unsigned integer. In either case, smaller values of the metric are preferred. Currently, this metric is only used to break ties between routes with equal preference from the same neighbor AS.

Internal BGP sessions carry at least one metric in the path attributes, which BGP calls the **`LocalPref`**. The size of the metric is identical to the MED.

For BGP versions 2 and 3, this metric is considered better when its value is smaller. For version 4, the metric is better when its value is larger. BGP version 4 sessions can optionally carry a second metric on internal sessions, this is an internal version of the Multi-Exit Discriminator. The use of these metrics depends on the type of internal protocol processing that is specified.

Originating a MED

MED is originated using the **`metricout`** field in three BGP statements:

– The **`group`** statement

– The **`peer`** statement

– The **`export`** statement

MED is imported using the **`med`** on the group keyword statement.

BGP group statement

Packets sent to this group of BGP peers have the MED in the BGP packet modified to be the MED value from this AS.

To use MEDs in routing computations, enter the following statement:

```
group type external peeras autonomous_system med
```

To send MEDs, enter the following statement:

```
group type routing peeras autonomous_system proto protocol
   interface interface_list metricout metric
```

BGP peer statement

Packets received for the BGP peer and aspath are also checked for MED within the BGP packet of this value by entering the following statement:

```
peer host metricout metric
```

BGP export statement

Packets exported to a BGP neighbor can select routes to be sent by specifying MED on the source BGP protocol by entering the following statement:

```
export proto bgp as 2750
        {
        proto bgp as 2704
        metric 10
                 {
                route-filter
                }
        }
```

For more information about BGP statements, see the BGP statement in chapter 3.

Configuring route reflection

Route reflection is a technique for efficiently communicating routes in an internal autonomous system (AS).

In an autonomous system, each BGP speaker peering with an external router is responsible for propagating reachability and path information to all other transit and border routers within that AS. This is typically done by establishing internal BGP connections to all transit and border routers in the local AS.

In autonomous systems with a large number of peers, this practice leads to a formidable mesh of TCP connections. Using internal BGP “route reflectors” reduces configuration, memory, and CPU requirements necessary to convey external route information to all other BGP peers in the AS. A router acting as a route reflector delivers (advertises) external routes between multiple client peers.

The internal peers of a route reflector are divided into client peers and non-client peers – a route reflector and its client peers form a cluster. Client peers need not be fully meshed and should not peer with internal speakers outside of their cluster. Non-client peers must be fully meshed.

Figure 1-4 shows how route reflection saves internal BGP (IBGP) advertising links by assigning a router reflector to deliver its external routes to peer routers:

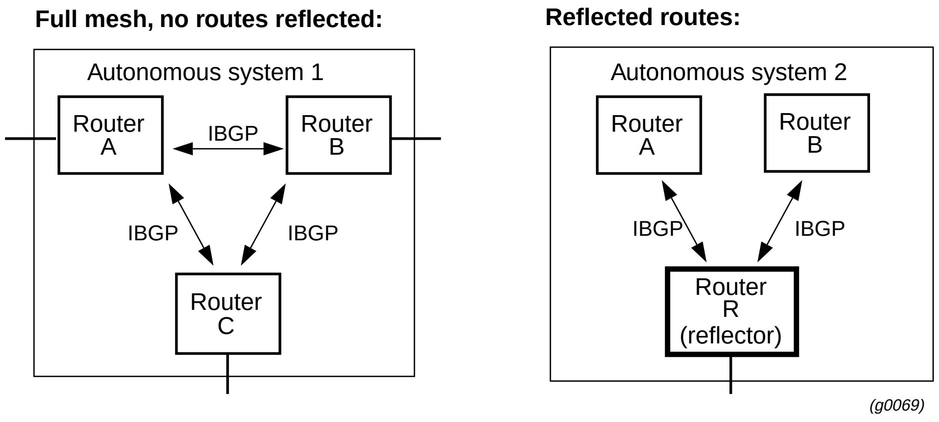

Figure 1-4. Full mesh and reflected route topologies

In autonomous system 1, all routers exchange external BGP information. In autonomous system 2, router R advertises external updates to client peer routers A and B as well as communicating A’s updates to B, and B’s to A. In both topologies, routers A and B have the same BGP tables.

Figure 1-5 illustrates what happens when two non-client peers that not participating in route reflection are added to the example shown in Figure 1-4.

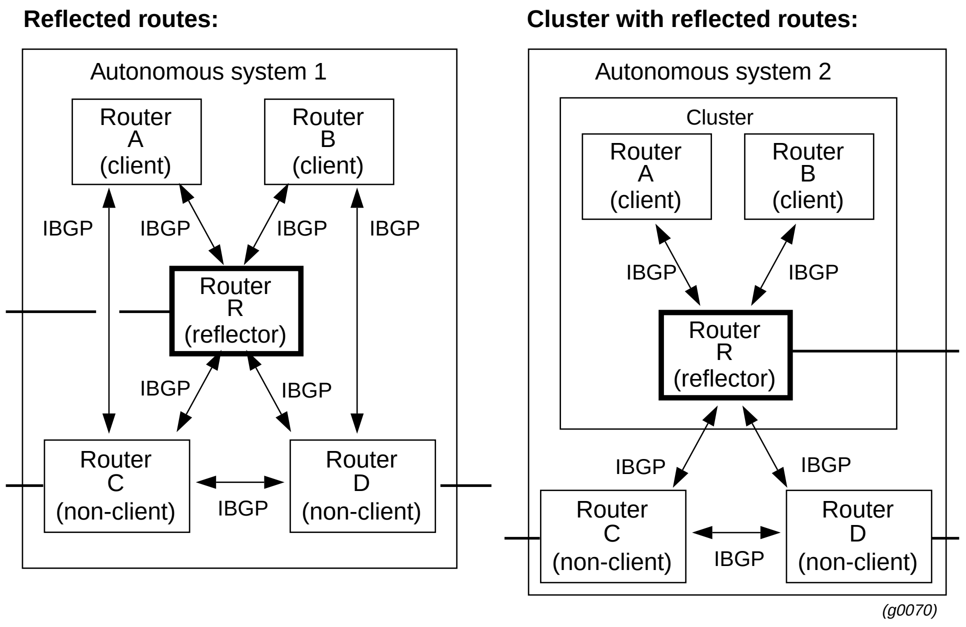

Figure 1-5. Client and non-client reflected route topologies

Router R reflects routes between these groups. Updates from a client router are sent to all other client and non-client routers. Updates from non-client routers C and D are sent to client routers A and B. Updates from an external peer router are sent to all client and non-client routers.

Route reflection example

The exampleillustrated in Figure 1-6 demonstrates how to configure `/etc/gated.conf` to establish route reflection for the set of routers illustrated here.

Routers A and B are clients of route reflector R. Together, routers A, B, and R form a route-reflection cluster. Routers C and D are in the same autonomous system as the other routers, but C and D are not part of the route reflection cluster.

The route reflector router must have an **`export`** statement; see **`reflector-client`** in chapter 3 for more information. In a real-world configuration, there may be more importation and exportation. In particular, there are no static routes or EBGP peers here. The EBGP peers would limit the effectiveness of the IBGP mesh.

The next several pages provide the GateD entries for each of the routers included in the route reflection example illustrated in Figure 1-6.

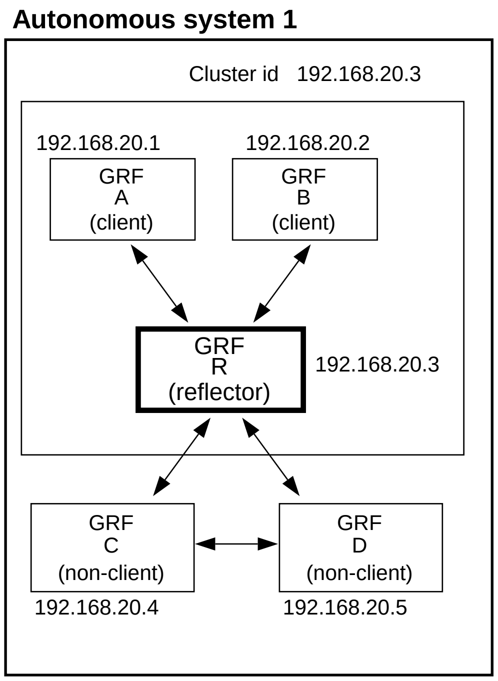

Figure 1-6. Route reflection configuration example

/etc/gated.conf entries for GRF A - a route reflector client

```
#  This is where the interfaces are defined
#
interfaces {
   interface all passive ;
   } ;

routerid 192.168.20.1 ;
autonomoussystem 1 ;

#
#  start BGP section
#  This shows only route reflector portion of the parameters.
#  You can add other parameters as needed.

bgp on {
        group type routing peeras 1 proto rip interface all
                {
                peer 192.168.20.3;
                };
         } ;
#
# End BGP section
#

rip yes;

#
# Import Policies
```

```
#
#End of File
#
```

/etc/gated.conf entries for GRF B - a route reflector client

```
# This is where the interfaces are defined
#
interfaces {
   interface all passive ;
   } ;

routerid 192.168.20.2 ;
autonomoussystem 1 ;

#
#  start BGP section
#
#  This shows only route reflector portion of the parameters.
#   You can add other parameters as needed.

bgp on {
        group type routing peeras 1 proto rip interface all
                {
                peer 192.168.20.3;
                };
         } ;
#
#  End BGP section
#

rip yes ;
#
# Import Policies
#

#End of File
```

/etc/gated.conf entries for GRF R

GRF R is a route reflector with clients A and B and non-clients D and E.

```
# This is where the interfaces are defined
#
interfaces {
   interface all passive ;
   } ;

routerid 192.168.20.3 ;
autonomoussystem 1 ;
#
#  start BGP section
#
#  This shows only route reflector portion of the parameters.
#  You can add other parameters as needed.
#
```

```
# When acting as a route reflector, it is necessary to export
# routes from the local AS into the local AS.

bgp on {
        clusterid 192.168.20.3 ;

        group type routing peeras 1 proto rip interface all reflec-
tor-client
                {
                peer 192.168.20.1;
                peer 192.168.20.2;
                };

        group type routing peeras 1 proto rip interface all
                {
                peer 192.168.20.4;
                peer 192.168.20.5;
                };
         } ;
#
# End BGP section
#

rip yes ;

#
# Export Policies
#
export proto bgp as 1 {
           proto bgp as 1 { all ;}
#End of File
```

/etc/gated.conf entries for GRF C - a non-client peer

```
# This is where the interfaces are defined
#

interfaces {
   interface all passive ;
   } ;

routerid 192.168.20.4 ;
autonomoussystem 1 ;

#
# start BGP section
#
bgp on {

        group type routing peeras 1 proto rip interface all
                {
                peer 192.168.20.3;
                peer 192.168.20.5;
                };
        } ;
#
#  End BGP section
#
rip yes ;

#
# Import Policies
#
#End of File
```

/etc/gated.conf entries for GRF D - a non-client peer

```
#  This is where the interfaces are defined
#

interfaces {
   interface all passive ;
   } ;

routerid 192.168.20.5 ;
autonomoussystem 1 ;

#
#  start BGP section
#
bgp on {

        group type routing  peeras 1 proto rip interface all
                {
                peer 192.168.20.3;
                peer 192.168.20.4;
                };
         } ;
#
#  End BGP section
#


rip yes;

#
# Import Policies
#
#  End of File
```

Introduction to OSPF

Open Shortest Path Routing (OSPF) is a shortest path first or link-state protocol. OSPF is an interior gateway protocol that distributes routing information between routers in a single autonomous system (AS). OSPF chooses the least cost path as the best path.

Suitable for complex networks with a large number of routers, OSPF provides equal cost multipath routing where packets to a single destination can be sent through more than one interface simultaneously. In a link-state protocol, each router maintains a database describing the entire AS topology that it builds out of the collected link state advertisements of all routers.

Each participating router distributes its local state, that is, the router's usable interfaces and reachable neighbors, throughout the AS by flooding. Each multi-access network that has at least two attached routers has a designated router and a backup designated router. The designated router floods a link state advertisement for the multi-access network and has other special responsibilities. The designated router concept reduces the number of adjacencies required on a multi-access network.

OSPF allows networks to be grouped into areas. Routing information passed between areas is abstracted, potentially allowing a significant reduction in routing traffic. OSPF uses four different types of routes, listed in order of preference:

- Intra-area
- Inter-area
- Type 1 external
- Type 2 external

Intra-area paths have destinations within the same area, inter-area paths have destinations in other OSPF areas and Autonomous System External (ASE) routes are routes to destinations external to the AS. Routes imported into OSPF as type 1 routes are supposed to be from IGPs whose external metrics are directly comparable to OSPF metrics. When a routing decision is being made, OSPF adds the internal cost to the AS Border router to the external metric. Type 2 ASEs are used for EGPs whose metrics are not comparable to OSPF metrics. In this case, only the internal OSPF cost to the AS Border router is used in the routing decision.

From the topology database, each router constructs a tree of the shortest paths with itself as the root. This shortest-path tree gives the route to each destination in the AS. Externally derived routing information appears on the tree as leaves. The link-state advertisement format distinguishes between information acquired from external sources and information acquired from internal routers, so there is no ambiguity about the source or reliability of routes.

Externally derived routing information is passed transparently through the autonomous system and is kept separate from OSPF's internally derived data. Each external route can also be tagged by the advertising router, enabling a passing of additional information between routers on the borders of the autonomous system.

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

OSPF intra- and inter-area routes are always imported into the GateD routing database with a precedence of 10. It would be a violation of the protocol if an OSPF router did not participate fully in the area's OSPF, so it is not possible to override this. Lucent recommends that you do not give other routes lower precedence values.

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

All routers in an area must agree on that area's parameters. A separate copy of the link-state algorithm is run for each area. Because of this, most configuration parameters are defined on a per area basis. All routers belonging to an area must agree on that area's configuration. Mis-configuration leads to adjacencies not forming between neighbors, and routing information might loop or not flow.

Authentication

All OSPF protocol exchanges are authenticated for security reasons. Authentication guarantees that routing information is only imported from trusted routers. A single method must be configured for each area. This enables some areas to use much stricter authentication than others.

There are two authentication methods available. The first uses a simple authentication key of up to 8 characters and is standardized. The second is still experimental and uses the MD5 algorithm and an authentication key of up to 16 characters.

The simple password provides very little protection because in many cases it is possible to easily capture packets from the network and learn the authentication key. The experimental MD5 algorithm provides much more protection as it does not include the authentication key in the packet.

The OSPF specification currently specifies that the authentication type be configured per area with the ability to configure separate passwords per interface. This has been extended to allow the configuration of different authentication types and keys per interface. In addition, it is possible to specify both a primary and a secondary authentication type and key on each interface. Outgoing packets use the primary authentication type, but incoming packets can match either the primary or secondary authentication type and key.

Introduction to RIP

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

A router running RIP in active mode broadcasts updates at set intervals. Each update contains paired values where each pair consists of an IP network address and an integer distance to that network. RIP uses a hop count metric to measure the distance to a destination. In the RIP metric, a router advertises directly connected networks at a metric of 1. Networks that are reachable through one other gateway are two hops, etc. Thus, the number of hops or hop count along a path from a given source to a given destination refers to the number of gateways that a datagram would encounter along that path. Using hop counts to calculate shortest paths does not always produce optimal results. For example, a path with hop count 3 that crosses three Ethernets can 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.

As delivered with most UNIX systems, RIP is run by the routing daemon, `routed` (pronounced route-d). A RIP routing daemon dynamically builds on information received through RIP updates. When started up, it issues a request for routing information and then listens for responses to the request. If a system configured to supply RIP hears the request, it responds with a response packet based on information in its routing database. The RESPONSE packet contains destination network addresses and the routing metric for each destination.

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

RIP version 2 (more commonly known as RIPv2) add additional capabilities to RIP. Some of these capabilities are compatible with RIPv1 and some are not. To avoid supplying information to RIPv1 routes that could be misinterpreted, RIPv2 can only use non-compatible features when its packets are multicast. On interfaces that are not capable of IP multicast, RIPv1-compatible packets are used that do not contain potentially confusing information.

Some of the most notable RIPv2 enhancements are described in the following sections.

Nexthop

RIPv2 can advertise a nexthop other than the router supplying the routing update. This is quite useful when advertising a static route to a dumb router that does not run RIP. It avoids having packets destined through the dumb router from having to cross a network twice.

RIPv1 routers ignore nexthop information in RIPv2 packets. This can result in packets crossing a network twice, which is exactly what happens with RIP I. So this information is provided in RIP I-compatible RIPv2 packets.

Network mask

RIPv1 assumes that all subnetworks of a given network have the same network mask. It uses this assumption to calculate the network masks for all routes received. This assumption prevents subnets with different netmasks from being included in RIP packets. RIPv2 adds the ability to specify the network mask with each network in a packet.

While RIPv1 routers ignore the network mask in RIPv2 packets, their calculation of the network mask can quite possibly be wrong. For this reason, RIP I-compatible RIPv2 packets must not contain networks that would be misinterpreted. These networks must only be provided in native RIPv2 packets that are multicast.

Authentication

RIPv2 packets can also contain one of two types of authentication strings that can be used to verify the validity of the supplied routing data. Authentication can be used in RIPv1-compatible RIPv2 packets, but be aware that RIPv1 routers ignore it.

The first method is a simple password in which an authentication key of up to 16 characters is included in the packet. If this does not match what is expected, the packet is discarded. This method provides very little security as it is possible to learn the authentication key by watching RIP packets.

The second method is still experimental and can change in incompatible ways in future releases. This method uses the MD5 algorithm to create a crypto-checksum of a RIP packet and an authentication key of up to 16 characters. The transmitted packet does not contain the authentication key itself, instead it contains a crypto-checksum, called the digest. The receiving router performs a calculation using the correct authentication key and discard the packet if the digest does not match. In addition, a sequence number is maintained to prevent the replay of older packets. This method provides a much stronger assurance that routing data originated from a router with a valid authentication key.

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

RIP I and network masks

RIP I derives the network mask of received networks and hosts from the network mask of the interface through the packet that was received. If a received network or host is on the same natural network as the interface over which it was received and that network is subnetted (the specified mask is more specific than the natural netmask), the subnet mask is applied to the destination. If bits outside the mask are set, it is assumed to be a host, otherwise it is assumed to be a subnet.

On point-to-point interfaces, the netmask is applied to the remote address. The netmask on these interfaces is ignored if it matches the natural network of the remote address or is all ones.

Unlike in previous releases, the zero subnet mask (a network that matches the natural network of the interface, but has a more specific, or longer, network mask) is ignored. If this is not desirable, a route filter can be used to reject it.

Protocol-precedence and preference

Protocol-precedence (which is usually referred to as precedence) is the value GateD uses to choose routes from one protocol over another. Preference is the value GateD uses to order the preference of routes within BGP. Precedence and preference can be set in the GateD configuration file in several different configuration statements: at the protocol level, at the interface level, or as part of the import/export policy. This overrides the default precedence assigned to each protocol.

The preference value is an arbitrarily-assigned value used to determine the order of routes to the same destination in a single routing database. Preference can be used to select routes from the same exterior gateway protocol (EGP) learned from different peers or autonomous systems. Each route has only one preference value associated with it, even though preference can be set at many places in the configuration file. The last or most specific preference value set for a route is the value used. (See the Glossary: Preference or Glossary: Precedence entries.) The lowest value is preferred. The range of preference values is 0 through 65536.

In the following example, the precedence applied to routes learned through RIP from gateway 138.66.12.1 is 75. RIP routes learned from gateways other than 138.66.12.1 receive a precedence of 90. The precedence set on the **`import`** statement overrides the precedence set in the **`rip`** statement. The precedence set on the **`interfaces`** statement applies only to the routes that are directly connected to that interface.

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

Note: Lucent recommends that changes to preference be used before changes to protocol-precedence. In other words, if you can resolve the issues using the preference within the routing protocol, do so. This method is preferred over changing the order of protocols through protocol-precedence. Also, note that precedence can be set in all places that preference can be set.

BGP route selection algorithm

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors. GateD uses the following criteria, in the order shown, to select the best path. If routes are equal at a given point in the selection process, then the next criterion is applied to break the tie:

1 Configured policy: the route with the best (numerically smallest) preference, as determined by the policy defined in `/etc/gated.conf`.

2 Local_Pref: the route with the highest local preference. Local_Pref is used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. Local_Pref can be set using the `localpref` option on the or `export` statements.

3 Shortest AS path: the route whose NLRI specifies the smallest set of destinations.

4 Origin IGP < EGP < incomplete: the route with an AS path origin of IGP is preferred. Next in preference is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.

5 MED (if not ignored): the route with the best multi-exit discriminator (MED) is preferred.

6 Shortest IGP distance: if both routes are from the same protocol and AS, the route with the lower metric is preferred.

7 Source IGP < EBGP < IBGP: prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.

8 Lowest router ID: the route whose nexthop is the lowest numeric IP address.

Assigning protocol-precedence

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

Table 1-1 summarizes the default precedence values for routes learned in various ways. The table lists the statements (some of these are clauses within statements) that set precedence, and shows the types of routes to which each statement applies. The default precedence for each type of route is listed, and the table notes precedence between protocols.

Table 1-1. Precedence values

Precedence of:	**Defined by statement:**	**Default value:**
Direct connected networks	interfaces	0
OSPF routes		10
Internally generated default	gendefault	20
Redirects	redirect	30
Routes learned through route socket	kernel	40
Static routes from config	static	60
RIP routes	rip	100
Point-to-point interface		110
Routes to interfaces that are down	interfaces	120
Aggregate/generate routes	aggregate/generate	130
OSPF AS external routes	ospf	150
BGP routes	bgp	170

Weighted route dampening

The basic idea of weighted route dampening is to treat routes that are being announced and withdrawn (flapping) at a rapid rate as unreachable. If a route flaps at a low rate, it should not be suppressed or suppressed only for a brief period of time. With weighted route dampening, 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 dampening are controlled by a few configurable parameters. For more information, see Weighted routed dampening statement in chapter 3.

GateD Configuration Tutorial

2

Chapter 2 provides a tutorial showing how to use dynamic routing in building a network.

Only Configuration A is available at this time. Other configuration examples will be added in future revisions of this manual

The following topics make up Configuration A:

Introduction to Configuration A 2-2

Descriptions of the configuration files 2-4

Configuration A topology. 2-8

Configuring and verifying interface connections 2-8

Configuring and verifying GateD. 2-13

Using GSM to verify /etc/gated.conf 2-16

Summary for Configuration A 2-20

.

Introduction to Configuration A

The complete tutorial is divided into five configurations, A through E. Currently, this chapter describes only Configuration A.

Topology divided into sub configurations

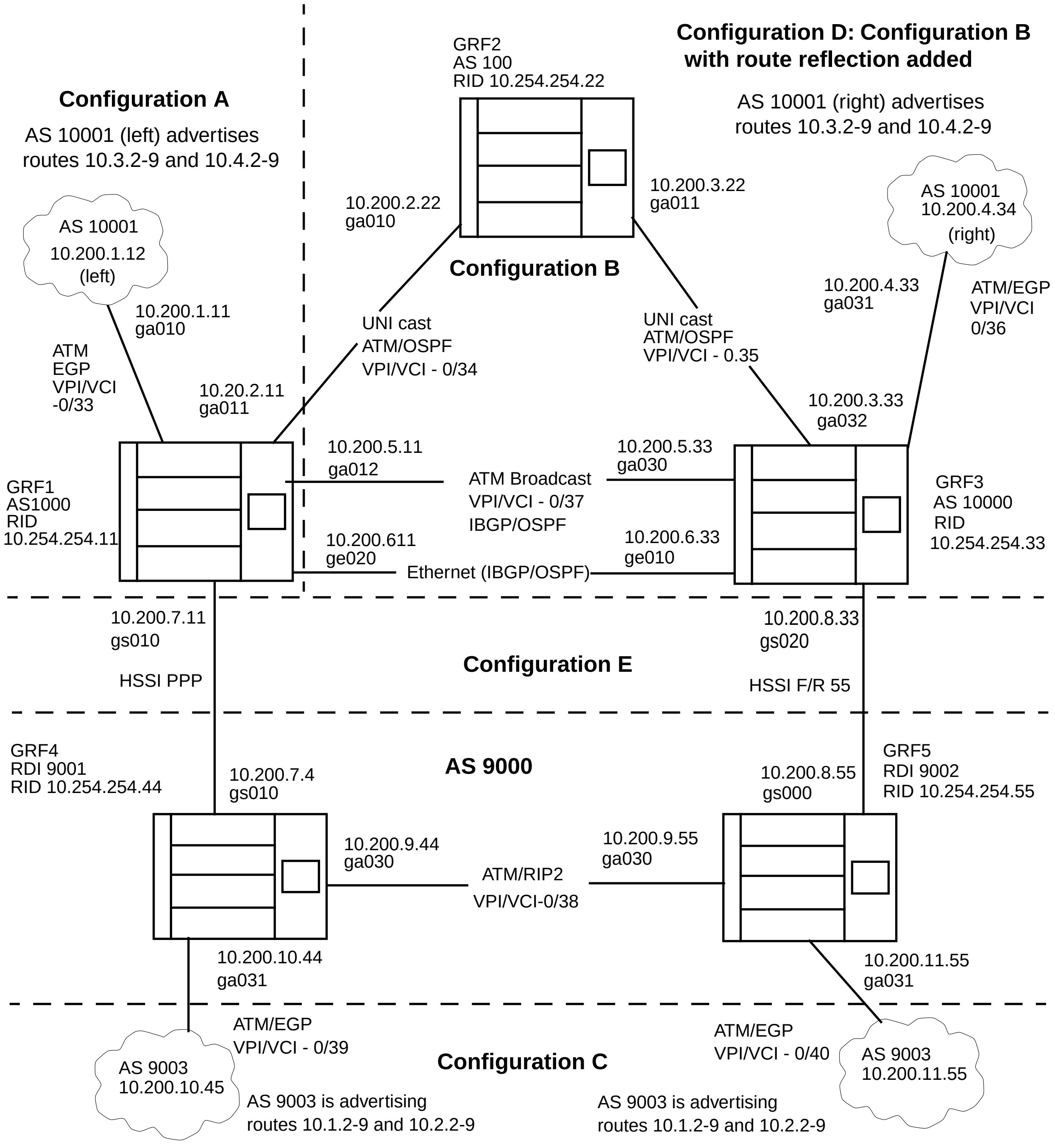

Figure 2-1. Planned configurations for GateD tutorial

Figure shows Configurations A through E, here is an overview of each:

1 Configuration A

A system connecting two ASs with a single pipe to the network. The ASs in this configuration are the left AS 10000 (GRF1) and the left AS 10001 cloud, and are connected through ATM/EBGP.

2 Configuration B

A system that grows to include two new routers, one of which has an additional peering session into AS 10001 (which is referred to as the right AS 10000 (GRF3) and right AS 10001 cloud). Configuration A is a subset of this new configuration. The three AS 10000 routers are fully meshed using OSPF/IBGP over ATM connections. The left and right AS 10000 have a redundant Ethernet connection using OSPF/IBGP. The third router in this configuration is referred to as the middle AS 10000 (GRF2).

3 Configuration D

A system that adds route reflecting into the configuration. This configuration combines Configurations A and B and adds route reflection. The middle router acts as the Route Reflector Server (RRS) AS 10000, while the right and left AS 10000 are Route Reflector Clients (RRCs). The RRS and RRCs are connected through OSPF/IBGP connections over an ATM connection. Communities are added to this configuration. Community tags are applied to the AS 10000 RRCs, while the AS 10000 RRS uses BGP preferences for routing.

4 Configuration C

The fourth configuration results in a confederated BGP system that is dual-homed. The configuration consists of a left RDI 9001 and right RDI 9002, connected through a direct ATM connection using RIP2, thus combining to form AS 9000. In addition, the left RDI 9001 is connected to the left AS 9003 cloud through a ATM/EBGP connection, while the right RDI 9002 is connected to the right AS 9003 cloud through the same type of connection. The second part of this configuration is to apply MED tags to the AS 9003s and set up the RDIs to accept MEDs.

5 Configuration E

The last step is to join configurations A through D into a one configuration example. This is referred to as configuration E. It shows how confederated ASs can talk to non-confederated ASs. RFI 9001 connects to the left AS 10000 through a HSSI PPP circuit while RDI 9002 connects to the right AS 10000 through a HSSI FR circuit.

Starting Configuration A

The purpose of Configuration A is to enable you to learn how to:

- configure the required files for an autonomous system (AS).
- configure routing protocols both within an AS and between ASs.

Lucent suggests you go through these steps to complete the tutorial:

1 Design the site topology. This can be done by simply drawing the topology shown in on a piece of paper.

2 Build the network in your lab, cable the systems in the network to each other. This step is site-specific, and is not detailed here.

3 Configure the logical connectivity of the network by editing the `/etc/grifconfig.conf` and `/etc/gratm.conf` files to provide the logical configurations for the network.

4 Define the protocols that are to be used on the network and establish the routing policy by editing the GateD `/etc/gated.conf` configuration file.

5 Verify peering sessions.

Descriptions of the configuration files

There are three files needed for Configuration A. They are:

- `/etc/grifconfig.conf`
- `/etc/gratm.conf`
- `/etc/gated.conf`

Each file has a unique purpose and works with the other configuration files to produce the network in this example. The following sections describe the basic purpose of each file.

Description of the /etc/grifconfig.conf file

The `/etc/grifconfig.conf` file specifies the IP addressing information for the networks attached to the GRF interfaces.

This includes interfaces on media cards and the directly-attached maintenance Ethernet interface (`de0` or `ef0`), and the mandatory `lo0` software loopback interface. The entries you make into this file are initiated and configured into the kernel when you reset the media card or when you have **grifconfig** re-read its configuration file.

The file has text that describes the entry syntax. The examples in this chapter show only the associated entries required in `/etc/grifconfig.conf`, not the syntax. Refer to Appendix C for a copy of the `/etc/grifconfig.conf` template containing the entry definitions. Here is the file's format:

```
# /etc/grifconfig.conf
# name   address     netmask         broad_dest   arguments
gs0386 192.168.10.10 255.255.255.0
```

The "Configuring System Parameters" chapter in the *GRF Configuration and Management* manual contains more information about using `/etc/grifconfig.conf`.

Interface name

The `ga0yz` interface name is required. An interface name describes an interface in terms of media type, chassis number, chassis slot, and logical interface number.

IP address

An Internet Protocol (IP) address is required. The Internet address is the 32-bit IP address for the logical interface being specified. The address is entered in standard dotted-decimal (octet) notation: `xxx.xxx.xxx.xxx.`

Netmask (optional)

A netmask determines which part of an IP address represents the network and which part represent the host machine. The default netmask is 255.255.255.0.

Broadcast / destination address (optional)

This address identifies a broadcast IP address for an Ethernet or FDDI interface or a destination address for a point-to-point ATM or HIPPI interface. Enter the broadcast or destination address in standard dotted-decimal (octet) notation. When a broadcast IP address is assigned to a logical interface, the netmask value is ignored. A dash (-) can be entered in the netmask column as a placeholder.

Argument field (optional)

The arguments field is currently used to specify MTU values on a per interface basis when the default MTU will not be used. If you want to use the arguments field and are not using a netmask or destination address, enter dashes as placeholders.

Description of /etc/gratm.conf file

The `/etc/gratm.conf` file defines the ATM circuits. Statements specify interfaces, PVCs, signaling protocols, ARP services, and traffic shapes. The entries you make into this file are initiated and configured into the kernel when you reset the media card or when you have **gratm** re-read its configuration file.

The file is lengthy and has text that describes the entry syntax. The examples in this chapter show only the associated entries required in `/etc/gratm.conf`, not the syntax.

The "ATM OC-3c Configuration Guide" chapter in the *GRF Configuration and Management* manual contains detailed information about using `/etc/gratm.conf`. Here is an excerpt:

Service section

ATM network services include ARP, local ATMARP server, and broadcast service. Give a different name to each type of service you define. These names are then assigned to the interfaces defined in the Interfaces section and specify the ATM service a logical interface will use or perform. There are three Service parameters: `name`, `type`, and `addr` (address).

```
Service name=value type=arp|bcast|arpserver addr=value \
[addr=value ...]
```

The text string `name` parameter identifies an instance of a service, for example, `arp0`, `arp1`, `broadcast_grp1`, `broadcast_grp2`, `arpserverA`, or `arpserverB`.

The `type` parameter specifies an ATM service and is either `arp`, `bcast`, or `arpserver`. ARP service returns ATM address information to the logical interface. Broadcast service enables the GRF to simulate broadcast over a logical IP network. ARP server enables the logical interface to function as a server for the attached ATM network.

The `addr` parameter relates to service type. An ARP service address is the NSAP or IP address of the remote server from which the interface obtains ATM address information. Broadcast values are IP addresses of hosts on the attached network to which copies of broadcast packets are sent. The `addr` for `type=arpserver` is the server NSAP address. The user can direct the system to determine the NSAP address with `addr=auto` or can manually enter the NSAP.

Traffic shaping section

In the Traffic Shaping section you define the available traffic shapes. There are two required parameters: `name` and `peak`, and three options, `sustain`, `burst`, and `qos`.

```
Traffic_Shape name=value peak=bps [sustain=bps burst=cells]  \
[qos=high|low]
```

Create a different text string `name` for each type of traffic shape you define. These names are assigned to the interfaces in the Interfaces and PVC sections, and specify resources allotted to a logical interface.

The `peak` parameter specifies peak cell rate in kilobits per second. The `sustain` (in kilobits per second) and `burst` (in cells) parameters are optional. If not specified, `sustain` and `burst` default to the peak rate you have specified. The Quality of Service `qos` parameter specifies whether the PVC will use high or low priority rate queues, it defaults to `qos=high`.

Signalling section

In the Signalling section you assign a signaling protocol to each physical interface, top and bottom. When switched virtual circuits are created on an interface, they will automatically use the protocol and other characteristics you have assigned that physical interface. Options enable you to change protocol, default mode, transmit clock, and per/circuit buffer queue settings.

```
Signalling card=hex connector=top|bottom   \
[protocol=UNI3.0|UNI3.1|NONE]  [mode=SDH|SONET] [clock=Ext|Int] \
[buf_limit=value]
```

There are two required parameters: `card` and `connector`, and four options, `protocol`, `mode`, `clock`, and `buf_limit`.

Use the slot number in hex for the `card` value. The physical `connector` value is either `top` or `bottom`. Protocol values are `UNI3.0`, `UNI3.1`, or `NONE` (PVCs require no protocol.). Specify `mode` to be either SDH or SONET, the default is SONET. The `clock` parameter is either external (`Ext`) or internal (`Int`), the default is internal.

The `buf_limit` parameter controls the depth of the queue of buffers on a given virtual circuit as part of a queue control feature, `buf_limit` defaults to 15 buffers. The queue control prevents a virtual circuit from consuming all the buffers on a card as the circuit becomes oversubscribed.

Interfaces section

In the Interfaces section you identify the logical interfaces configured on the ATM card.

```
Interface ifname [traffic_shape=shape_name][service=service_name]\
       [bridge_method=method [,restriction]]
```

There is one required parameter, the `ga0yz ifname`, and three options, `service`, `traffic_shape`, and `bridge_method`. The `traffic_shape=` parameter must follow the service= parameter or the file does not parse correctly. Identify the interface with the ga0yz interface name. Use a definition from the Service section for service_name. Use a definition from the Traffic Shaping section for shape_name.

PVC section

In the PVC section you assign three required parameters to each permanent virtual circuit: the interface name (ga0yz), a VPI/VCI, and a protocol.

```
PVC ifname VPI/VCI
proto=ip|raw|vc|ipnllc|isis|llc[,bridging] \
|vcmux_bridge,bpro|vc_atmp|llc_atmp       [input_aal=3|5|NONE] \
[traffic_shape=shape]   [dest_if=logical_if [dest_vc=VPI/VCI]]
```

The ga0yz interface name and the VPI/VCI parameters locate the virtual circuit. Although many protocol options are listed, not all are available. Refer to the "ATM OC-3c Configuration Guide" chapter in the *GRF Configuration and Management* manual for a list of protocols currently available on the ATM OC-3c media card.

The traffic_shape= parameter must be one of the name= entries defined in the Traffic Shaping section. If you do not specify a traffic shape, this parameter defaults to a shape of 155000 kbps and a high Quality of Service (qos=high).

The destination interface and destination VPI/VCI are used only if you specify proto=raw. The dest_if parameter specifies the gx0yz name of the destination GRF interface for this raw adaptation layer connection. The dest_vc parameter specifies the VPI/VCI for that destination interface.

Description of /etc/gated.conf

You define the protocols that are to be used on your network, the interfaces and ASs that are to participate, and set dynamic routing policy in the /etc/gated.conf file. Routing policy is also referred to as the import/export policy. Import statements enables a way to learn about other routes, while export statements enable a way to advertise or redistribute routes.

The file consists of a sequence of configuration statements terminated by a semicolon (;). An out-of-order statement causes an error when the /etc/gated.conf file is parsed. The statements, in correct order, are as follows:

- Options statements
- Interface statements
- Definition statements
- Protocol statements
- Static statements
- Control statements
- Aggregate statements

Definitions for each statement and their parameters are in Chapter 3 of this manual, "GateD Configuration Statements."

Configuration A topology

Configuration A is a simple network topology connecting two autonomous systems, AS 10000 and AS 10001.

AS 10000 is a GRF (GRF1) and AS 10001 is a separately-maintained external AS. AS 10001 advertises destination IP networks 10.3.2 - 9 and 10.4.2 - 9. Configuration A has a single network connection using EBGP as the routing protocol between the two ASs. The GRF1 connection is over an ATM OC-3c media card. Assume that both sides of this configuration support ATM inverse ARP. GRF supports ATM inverse ARP by default.

The following logical IP interface addresses are used in configuration A:

```
GRF1: 10.200.1.11
AS 10001: 10.200.1.12
```

The following router ID (RID) is used for configuration A:

```
GRF1: 10.254.254.11
```

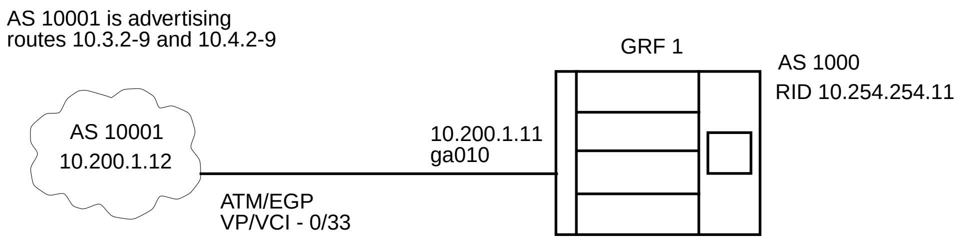

Figure 2-2. Topology of Configuration A

Configuring and verifying interface connections

To configure the logical connections for configuration A, edit these files:

- `/etc/grifconfig.conf`
- `/etc/gratm.conf`

After you define the necessary fields in the configuration files, you will execute commands to bring up the interface and other commands to test the logical connections. These files and commands are explained in the following sections.

Editing the /etc/grifconfig.conf file

Here are the entries in `/etc/grifconfig.conf` for configuration A. Each is explained below:

```
# /etc/grifconfig.conf
# name   address     netmask         broad_dest   arguments
de0 206.146.165.2 255.255.255.0        # maintenance Ethernet interface
lo0 127.0.0.1 255.0.0.0                #  standard loopback interface
lo0 10.254.254.11 255.255.255.255      # loopback for RID (always active)
```

```
   ga010 10.200.1.11 255.255.255.0          # ATM interface to AS 10001
```

The first two entries define the interface name, IP address, and netmask for the control board maintenance Ethernet interface and the standard loopback interface, respectively. The `de0` information is assigned by the site network administrator; the `lo0` entry is a mandatory entry. If `lo0` is not defined, GateD aborts.

The third entry defines the interface name, IP address, and netmask for the loopback or secondary alias for the GRF1 router ID (RID). In this, and following examples, the address is a class A address in the restricted range, so it is a real address though not one assigned to any live internet participant.

This entry is a logical interface address that is not associated with any physical interface (as is, for example, the **`ga010`** entry). So if one or more of the physical components fails, for example, the media board or the actual network connection, a router in that AS may benefit by an alternate path, which is what the RID loopback path provides. The importance of this entry will become more evident in later configurations (for example, OSPF uses the loopback alias instead of the second entry, because the second entry may not always be available).

The fourth entry defines the direct interface address of the ATM card in the GRF system. For this interface, the first character, `g` must always be there to indicate this is a GRF interface. The second character, `a` indicates media hardware, in this case an ATM card. The third character must be 0. The fourth character, `1` indicates the GRF slot number in which the ATM card is located. The fifth character, `0` specifies the number of the logical interface on the ATM card.

Refer to the comments in the `/etc/grifconfig.conf` file for more information about defining physical and logical interfaces for the media card.

Editing the /etc/gratm.conf file

For configuration A, the `/etc/gratm.conf` file is used to set up a virtual circuit, in this case a permanent virtual circuit (PVC).

The non-commented lines are the entries in `/etc/gratm.conf` for configuration A. Explanations for the entries follow the example.

```
# ATM traffic shape:
# peak=peak cell rate, sustain=sustained cell rate, burst=burst size
(in cells)
# Each peak rate specified, automatically creates a rate queue.
#
Traffic_shape name=high_speed_high_quality peak=155000 \
sustain=155000 burst=2048 qos=high          # Use entire OC-3 bandwidth.
# Signalling:
# The default setting for protocol is NONE and is not needed when
# configuring a PVC.
Signalling card=1 connector=top protocol=NONE
                                            # Signaling is not # required for PVCs.
# Interface:
```

```
# Interface ifname [service=service_name] [traffic_shape=shape_name]
# service and traffic_shape are optional
Interface ga010                        # GRF interface to AS 10001.
#
# PVC:
# PVC is where you map your logical interface to the VPI/VCI pair
# and specify traffic shaping.
PVC ga010 0/33 proto=ip traffic_shape=high_speed_high_quality
```

First, define the circuit's bandwidth resources in the ATM traffic shape section. The traffic shape name (`high_speed_high_quality`) reflects the maximum values assigned to `peak`, `sustain`, `burst`, and `qos` parameters. The shape name is used later in the PVC section to assign the resources.

The Signalling section enables switched virtual circuits (SVCs). Because only PVCs are being used, and signaling is not needed for PVCs, signaling is turned off. The `Signalling card=1 connector=top protocol=NONE` entry in the Signalling section turns off signaling.

The Interface section defines the logical ATM interface in the GRF to which the PVC is mapped. In this case, the PVC is mapped to the `ga010` logical interface of the ATM board with the `Interface ga010` entry.

Each virtual circuit is represented by a pair of numbers consisting of the virtual path identifier (VPI) and the virtual circuit identifier (VCI). The pair is separated by a slash (/). The VPI/VCI pair is assigned to the logical ATM interface in the PVC statement.

In this example, the `PVC ga010 0/33 proto=ip traffic_shape=high_speed_high_quality` entry specifies the following:

A PVC is assigned to logical interface `ga010` on VPI `0` and VCI `33`. This circuit runs the IP protocol. The `traffic_shape=high_speed_high_quality` specifies that the PVC has the full bandwidth of the ATM OC-3 card.

Verifying logical and physical connections

These three steps initialize and verify the interface configuration and connectivity:

- Execute the **`gratm`** command.
- Execute the **`grifconfig -v`** command.
- Execute **`ifconfig`**, **`grarp`**, and **`ping`** commands to verify the connections.

1 Execute the **`gratm`** command to send the contents of the `/etc/gratm.conf` file to the media card:

```
# gratm ga01
gratm: Begin on-the-fly PVC configuration for card 0x1
gratm: Sent 32 grinches for card 0x1
```

This **`gratm`** command operates at the media card level because the logical interface address is not using the fifth or sixth characters which specify the number of the logical interface.
If you use the whole address with the **`gratm`** command, you get these results:

```
# gratm ga010
gratm: Configuring an interface is no longer supported on ATM/Q.
Use gratm ga01 instead
```

2 Execute the **`grifconfig -v`** command; the **`-v`** verbose option sends the results of the command to standard output. This command initializes the configuration of the GRF network interfaces.

```
# grifconfig -v
de0 - adding/modifying inet 206.146.165.2
grifconfig: adding route, network ‘206.146.165.0’, address
‘206.146.165.2’
lo0 - adding/modifying inet 127.0.0.1
grifconfig: adding route, network ‘127.0.0.0’, address ‘127.0.0.1’
lo0 - adding/modifying inet 10.254.254.11
grifconfig: adding route, network ‘10.254.254.11’, address
‘10.254.254.11’
ga010 - adding/modifying inet 10.200.1.11
grifconfig: adding route, network ‘10.200.1.0’, address
‘10.200.1.11’
```

The result of executing the **`gratm`** and **`grifconfig`** commands is that the physical and logical connections are now initialized and ready for use. At this point, you should verify your configuration. You can do this by executing a series of commands, as shown in the following paragraphs.

3 Execute the **`ifconfig -au`** command, as shown in the following example:

```
# ifconfig -au
de0: ether flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,
   MULTICAST> inet 206.146.165.2 netmask 0xffffff00
   broadcast 206.146.165.255
rmb0: ether flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,
   SIMPLEX,MULTICAST> grit 0:0x40:0 broadcast 0:0x41:255
lo0: loop flags=8009<UP,LOOPBACK,MULTICAST>
   inet 127.0.0.1 netmask 0xff000000
   grit 0:0x48:0
ga010: gritatm flags=b043<UP,BROADCAST,RUNNING,LINK0,LINK1,
   MULTICAST> inet 10.200.1.11 netmask 0xffffff00
   broadcast 10.200.1.255
```

The results of this command show that the interfaces are up (`UP`) and running (`RUNNING`). If you do not see the UP and RUNNING keywords, then something is wrong with your configuration. In addition, the `LINK0` and `LINK1` status flags in the `ga010` entry are important. They indicate that both the logical (LINK0) and physical (LINK1) links are available.

Next, verify that the inverse ARP is working by executing the **`grarp -a`** command:

```
# grarp -a
nw-passage.customer.com (206.146.165.1) at 0:c0:80:1b:70:9b
GRF1.customer.com (206.146.165.2) at 0:c0:80:82:85:be permanent
leaded.customer.com (206.146.165.14) at 8:0:69:b:b6:57
ga010   (10): 10.200.1.12 at VPI=0, VCI=33 permanent
```

Finally, use the **ping** command on both sides of your network, as shown in the following example. If data is returned, you know that the connections are up and working.

```
# ping 10.200.1.11
ICMP ECHO 10.200.1.11 (10.200.1.11):  56 data bytes
64 bytes from 10.200.1.11: icmp_seq=0 ttl=255 time=0.720 ms
64 bytes from 10.200.1.11: icmp_seq=1 ttl=255 time=0.631 ms
64 bytes from 10.200.1.11: icmp_seq=2 ttl=255 time=0.620 ms
64 bytes from 10.200.1.11: icmp_seq=3 ttl=255 time=0.617 ms

--- 10.200.1.11 ICMP ECHO statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.617/0.640/0.720 ms

# ping 10.200.1.12
ICMP ECHO 10.200.1.12 (10.200.1.12):  56 data bytes
64 bytes from 10.200.1.12: icmp_seq=0 ttl=255 time=1.030 ms
64 bytes from 10.200.1.12: icmp_seq=1 ttl=255 time=0.782 ms
64 bytes from 10.200.1.12: icmp_seq=2 ttl=255 time=0.779 ms
64 bytes from 10.200.1.12: icmp_seq=3 ttl=255 time=0.799 ms

--- 10.200.1.12 ICMP ECHO statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.779/0.840/1.030 ms
```

Configuring and verifying GateD

The next step is to configure GateD by editing the `/etc/gated.conf` file to define routing protocols and to establish routing policy. Then verify the GateD configuration.

Editing the /etc/gated.conf file

The following example shows the entire `/etc/gated.conf` file for configuration A.

Unlike the `/etc/grifconfig.conf` and `/etc/gratm.conf` files, the `/etc/gated.conf` file does not include commented text to explain how to use the different parts of the file. The following paragraphs explain each group of statements in more detail.

```
# Interface statements: defines GateD interfaces.
interfaces {
  interface all passive;     #Maintain same preference regardless  of interface status
};

# Definition statements: general config statements that relate to all of GateD
# or at least to more than one protocol.
routerid 10.254.254.11;                    # RID for GRF1
autonomoussystem 10000;

#  Protocol statements: defines the protocols for the router.
rip no;     # RIP is off by default in 1.3.[7-8]?

bgp on {
  traceoptions “/var/tmp/gated_bgp”
  replace size 200k files 2 all;

  group type external peeras 10001 {    # Group type external means EBGP
    peer 10.200.1.12;
  };
};

# Control statements: controls routes that are imported from peers and routes
#  that are exported to peers.
import proto bgp as 10001 { all; };     # Importing all BGP routes from 10001

export proto bgp as 10001 {              # Exporting all AS 10000 routes into
                                          #BGP toward 10001
  proto bgp as 10000 { all; };
};
```

In the Interface statements, the **`interface all passive`** statement tells GateD to maintain the same preference even if it believes it is not functioning properly due to a lack of received routing information.

The Definition statement does two things. First, it establishes the router ID for GRF1 (through **`routerid 10.254.254.11`**) for use by BGP (and OSPF, though it is not used in this configuration). Second, it sets the autonomous system number of the GRF1 router (through **`autonomoussystem 10000`**). This is a required statement, since the configuration uses BGP as one of its routing protocols.

In the Protocol statement, two components are used. The first one, `rip no`, ensures RIP is not running. In previous releases of GRF, RIP was off by default. Beginning in the 1.4.6 release, the default is on, so this statement turns it off.

The next component (**`bgp on`**) turns on BGP, the protocol that the GRF1 router will be using to communicate with AS 10001. The **`traceoptions`** component sets up a file to log all activity for GateD. Error and normal operation messages are sent to `/var/tmp/gated_bgp`. The **`replace size 200k files 2 all`** statement says that when the file is greater than 200k, trim the file, and start again. The `2` in this statement says that two copies of the log file can be present at any time. The `/var/tmp/gated_bgp` file contains the current log of activity, while the second file contains the previous 200ks of information. These files can be used for debugging purposes.

The next statement component is still part of the BGP protocol definition. The **`group type external peeras 10001`** statement component establishes the peering session for GRF1, which for configuration A is with AS 10001 (IP address **`10.200.1.12`**). The key words in this statement are **`type external`**. They mean this peering session is using EGP. The **`peeras 10001`** specifies that the peering session using EGP is with AS 10001. The **`peer 10.200.1.12`** statement defines the interface address of the AS 10001 peer. For EGP (an exterior routing protocol), you must use the IP address. Configurations that use interior routing protocols require the router ID to be used.

The last statement group is the Control statements. This part of the file establishes the routing policy for GateD on GRF1. The Control statements define the routes that are imported from GRF1's peers and those routes that GRF1 can export to its peers.

Basically, the **`import`** statement allows GateD on GRF1 to learn all the routes that are being advertised by its selected peer(s). The **`export`** statement allows GateD on GRF1 to advertise all the routes it knows about to its selected peers. Without the **`import`** control statement, GateD implicitly accepts all routing information.

The **`import proto bgp as 10001 { all; }`** statement says to import all BGP routes from AS 10001 to AS 10000.

The **`export proto bgp as 10001`** statement says to export all AS 10000 BGP routes to AS 10001. The second part of the **`export`** statement is the export list (that is, **`proto bgp as 10000 { all; }`**). The export list specifies the export based on the origin of the route (the syntax varies, depending on the source). For configuration A, the export list specifies that the source routes are all the BGP routes known by AS 10000.

Testing and saving the /etc/gated.conf file

After editing the `/etc/gated.conf` file, you should execute the following commands to ensure that the file parses correctly and that GateD is running:

1 Parse the `/etc/gated.conf` file and check for errors.

Do this by running the **`gdc`** checkconf command (you can also use the **`gated -C`** command). The **`gdc checkconf`** command writes any errors to the `/var/tmp/gated_parse` file, while the **`gated -C`** command writes the errors to standard output. If errors are reported, you will have to re-edit the file. Parse the file until no errors are reported.

2 Check the status of GateD.

Ensure it is running by executing the `gdc running` command.

If it is running, then you can execute the **`gdc reconfig`** command to pick up changes in the GateD configuration file.

If GateD is not running, then execute the **`gdc start`** command.

Using GSM to verify /etc/gated.conf

Verifying the peering session

The GateD State Monitor (GSM) utility provides an interactive interface to a running GateD daemon. You can use it to query internal GateD variables and check the status of GateD. See Chapter 4, “GateD State Monitor,” for information about starting GSM.

The following examples show some of the information that can be obtained by using GSM.

The **`show bgp summary`** command returns a short summary of the BGP peering sessions. It shows that one neighbor, whose IP address is `10.200.1.12`, is also AS 10001.

```
GateD-GRF1.customer.com> show bgp summary
Neighbor        V AS    Est.# Est(s) #routes #active #to   MsgRx
MsgTx   State
10.200.1.12    4 10001 6    9      16      16      8    3     9
Established
BGP summary, 1 groups, 1 peers
```

Determining BGP status

The **`show bgp detail`** command returns a detailed report of the state of BGP on GRF1. The commented lines (called out with a #) explain output in this example. You do not see these commented lines in the output when you execute the command.

```
GateD-GRF1.customer.com> show bgp detail prx 10001 10.200.1.12 0/0
```

# the syntax of this command says to show BGP details received from
#AS 10001 via the remote interface 10.200.1.12 for all routes

```
group type External AS 10001 local 10000 Flags
  Peer: 10.200.1.12 ID: 10.200.1.11 Version: 4  Gateway: (null)
  Peer is reachable through interface: 10.200.1.11  (ga010)
```

#Shows the local (AS 10000) interface that is connected through to AS 10001

```
  Local ID: 10.254.254.11
```

#Shows the router ID for GRF1

```
  Local Addr:
                10.200.1.11+179
  Local port 179  remote port 2086
```

#Shows the BGP TCP connection on AS 10000/GRF1 and the remote
#BGP connection on AS 10001, respectively.

```
  Flags 0x820
  State 0x6
  Established Transitions 5 Established Time 1298
  LastSt: OpenConfirm LastEv: RecvKeepAlive LastErr: None
```

#The previous two lines show that an open connection has been
#established. The “LastSt” indicates the last state of the connection,
#while “LastEv” indicates the last event on the connection.

```
     Options 0x0
     MED exported -1
     Proto-precedence/BGP preference 170/-
     Number of Notifications from this peer: 0,
     Number of Notifications sent to     peer 0
     Number of routes from this peer: 16
     Number of active routes from this peer: 16
```

#Shows the number of routes imported.

```
     Number of routes exported to this peer: 8
```

#Shows the number of routes exported.

```
     Messages in 30 (updates 2, not updates 28) 700 octets
     Messages out 32 (updates 4, not updates 28) 739 octets
     Last traffic (seconds): Received 46, Sent 4, Checked 21
     Time since last Keepalive sent: 4 seconds
```

#This line is also shown as the last event to have occurred, as shown in
#line above that starts with “LastSt:”.

```
     Time since last Update Recv’d: 1298 seconds
     Max Update Tx per second: unlimited
     Inbound Timer: 0
     Outbound Timer: 0
     Received and buffered Octets: 0
     Active Holdtime: 180
     Route Queue Timer:
       unset
     Route Queue: empty

    Define symbols: * = active route + best BGP route
                    + = active route -- not BGP route
                    ^ = best BGP route (but not active)
                    - = not best BGP route (not active)
                    @ = suppressed BGP route
                    | = not BGP, or not avail. for selection
     Route    Mask       NextHop      Prec/Pref Metric/2   Tag  ASPath
   *  10.3.2    255.255.255  10.200.1.12     170/-     -1/100   0
      (10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
      ID 10.200.1.11
```

#The * shows the route is active; it must be active if EBGP is configured
#correctly. “10.200.1.12” is the next hop to get to 10.3.2 and is the
#interface of the BGP peer. The “(10000) 10001 1111 2222 3333 IGP” is the AS
#path to get to 10.3.2 (that is, through AS 10000 and then to AS 10001;
#the“1111, 2222, 3333” notation is specific to how this example was generated.
#The “orig NH 10.200.1.12 ID 10.200.1.11” shows the route originated from
#the next hope (the IP followed by the RID).

```
 *  10.3.3        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11
```

```
*  10.3.4        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.3.5        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.3.6        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.3.7        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.3.8        255.255.255    10.200.1.12       170/-   -1/100   0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.3.9        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.2        255.255.255    10.200.1.12        170/-   -1/100  0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.3        255.255.255     10.200.1.12      170/-    -1/100  0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.4        255.255.255     10.200.1.12       170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.5        255.255.255     10.200.1.12       170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.6        255.255.255     10.200.1.12       170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.7        255.255.255     10.200.1.12       170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11
```

```
*  10.4.8        255.255.255     10.200.1.12       170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11

*  10.4.9        255.255.255     10.200.1.12      170/-    -1/100   0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
 ID 10.200.1.11
```

Summary for Configuration A

Configuration A shows how to configure a simple network connection between two autonomous systems over ATM OC-3 media using BGP as the routing protocol. It also demonstrates how to:

- Configure and bring up the logical connections between the two ASs by editing the `/etc/grifconfig.conf` file.
- Create VPI/VCI pairs and assign them as a permanent virtual circuit (PVC) by editing and activating the `/etc/gratm.conf` file.
- Define GateD protocols and establish a routing policy for GateD by editing, checking, and activating the `/etc/gated.conf` file.

  It is important to remember that without the import/export statements, GateD does not know how to receive advertised routes from other ASs or how to advertise what it knows about routes to other ASs.
- Use the GateD State Monitor (GSM) to verify the peering session from a brief summary and from more detailed output.

GateD Configuration Statements

3

Chapter 3 describes the following statements and related topics:

Syntax of GateD configuration file 3-2
Statement summary 3-3
Protocol-precedence and preference 3-4
Trace statement 3-6
Directive statement. 3-8
Options statement. 3-8
GSM statement. 3-9
Interface statement 3-12
Definition statement 3-17
RIP statement 3-19
OSPF statement 3-22
BGP statement 3-28
Weighted route dampening statement 3-40
ICMP statement 3-41
Router discovery protocol 3-42
Router discovery server statement 3-43
Router discovery client statement 3-45
Kernel statement. 3-47
Static statement 3-51
Control statements 3-53
Route filtering 3-53
Matching AS paths 3-57
Import statement 3-64
Export statement 3-68
Route aggregation and generation statements 3-75

Syntax of GateD configuration file

The GateD configuration file, `/etc/gated.conf`, consists of a series of statements. A semicolon (`;`) terminates a segment and separates statements.

Statements are composed of tokens separated by white space that can be any combination of blanks, tabs, and new lines. This structure identifies the parts of the configuration associated with each other and with specific protocols.

Comments can be specified in either of two forms:

- beginning with a pound sign (#) and running to the end of the line
- using C style, starting with a `/*` and continuing until it reaches `*/`

Syntax description conventions

In this manual, keywords and special characters that the parser expects exactly as shown appear in **`bold`** monospaced font. Variable parameters are shown in *`italic`* monospaced font. Square brackets ([and]) are used to show optional keywords and parameters. A vertical bar (|) separates optional parameters. Parentheses ( ) group keywords and parameters, when necessary.

For example, in the following syntax example, the square brackets indicate that the parameters are optional. The keywords are **`backbone`** and **`area`**. The vertical bar indicates that either **`backbone`** or **`area`** *`area`* can be specified. Because *`area`* is in italic font, it is a variable parameter provided by you. The parentheses group a keyword and its variable parameter.

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

Order of statements

Entering a statement out of order causes an error when `/etc/gated.conf` is parsed. The type of configuration statements and the order in which these statements appear in `/etc/gated.conf` are as follows:

- options statements
- GSM statements
- interface statements
- definition statements
- protocol statements
- static statements
- control statements
- aggregate statements

Two types of statements do not fit in these categories: %directive statements and %trace statements. These statements provide instructions to the parser and control tracing from`/etc/gated.conf`. They do not relate to the configuration of any protocol and can appear anywhere in the `/etc/gated.conf` file.

Statement summary

Table 3-1 lists each statement name and type, and gives a synopsis of the statement's function.

Table 3-1. Summary of GateD configuration statements

Statement	Type	Function
%directory	(directive)	Sets the directory for include files.
%include	(directive)	Includes a file into `/etc/gated.conf`.
traceoptions	(trace)	Specifies which events are traced.
options	(definition)	Defines GateD options.
gsm	(definition)	Defines GSM options.
interfaces	(definition)	Defines GateD interfaces.
autonomoussystem	(definition)	Defines the AS number.
multipath	(definition)	Defines the installation of multiple gateways.
routerid	(definition)	Defines the originating router (BGP, OSPF).
martians	(definition)	Defines invalid destination addresses.
rip	(protocol)	Enables RIP protocol.
kernel	(protocol)	Configures kernel interface options.
ospf	(protocol)	Enables OSPF protocol.
bgp	(protocol)	Enables BGP protocol.
routerdiscovery	(protocol)	Enables Router Discovery protocol.
redirect	(protocol)	Configures the processing of ICMP redirects.
weighted route dampening	(protocol)	Minimizes effects of route flapping
icmp	(protocol)	Configures the processing of general ICMP packets.
static	(static)	Defines static routes.
import	(control)	Defines which routes to import.
export	(control)	Defines which routes to export.
aggregate	(control)	Defines which routes to aggregate.
generate	(control)	Defines which routes to generate.

The rest of this chapter provides detailed definitions and descriptions of each of the GateD statements.

Protocol-precedence and preference

Protocol-precedence (which is usually referred to as precedence) is the value GateD uses to choose routes from one protocol over another. Preference is the value GateD uses to order the preference of routes within BGP. Precedence and preference can be set in the GateD configuration file in several different configuration statements: at the protocol level, at the interface level, or as part of the import/export policy. This overrides the default precedence assigned to each protocol.

The preference value is an arbitrarily-assigned value used to determine the order of routes to the same destination in a single routing database. Preference can be used to select routes from the same exterior gateway protocol (EGP) learned from different peers or autonomous systems. Each route has only one preference value associated with it, even though preference can be set at many places in the configuration file. The last or most specific preference value set for a route is the value used. (See the Glossary: Preference or Glossary: Precedence entries.) The lowest value is preferred. The range of preference values is 0 through 65536.

In the following example, the precedence applied to routes learned through RIP from gateway 138.66.12.1 is 75. RIP routes learned from gateways other than 138.66.12.1 receive a precedence of 90. The precedence set on the **`import`** statement overrides the precedence set in the **`rip`** statement. The precedence set on the **`interfaces`** statement applies only to the routes that are directly connected to that interface.

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

Note: Lucent recommends that changes to preference be used before changes to protocol-precedence. In other words, if you can resolve the issues using the preference within the routing protocol, do so. This method is preferred over changing the order of protocols through protocol-precedence. Also, note that precedence can be set in all places that preference can be set.

BGP route selection algorithm

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors. GateD uses the following criteria, in the order shown, to select the best path. If routes are equal at a given point in the selection process, then the next criterion is applied to break the tie:

1 Configured policy: the route with the best (numerically smallest) preference, as determined by the policy defined in `/etc/gated.conf`.

2 Local_Pref: the route with the highest local preference. Local_Pref is used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. Local_Pref can be set using the `localpref` option on the `import` or `export` statements.

3 Shortest AS path: the route whose NLRI specifies the smallest set of destinations.

4 Origin IGP < EGP < incomplete: the route with an AS path origin of IGP is preferred. Next in preference is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.

5 MED (if not ignored): the route with the best multi-exit discriminator (MED) is preferred.

6 Shortest IGP distance: if both routes are from the same protocol and AS, the route with the lower metric is preferred.

7 Source IGP < EBGP < IBGP: prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.

8 Lowest router ID: the route whose nexthop is the lowest numeric IP address.

Assigning protocol-precedence

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

Table 3-2 summarizes the default precedence values for routes learned in various ways. The table lists the statements (some of these are clauses within statements) that set precedence, and shows the types of routes to which each statement applies. The default precedence for each type of route is listed, and the table notes precedence between protocols.

Table 3-2. Precedence values

Precedence of:	**Defined by statement:**	**Default value:**
Direct connected networks	interfaces	0
OSPF routes		10
Internally generated default	gendefault	20
Redirects	redirect	30
Routes learned through route socket	kernel	40
Static routes from config	static	60
RIP routes	rip	100
Point-to-point interface		110
Routes to interfaces that are down	interfaces	120
Aggregate/generate routes	aggregate/generate	130
OSPF AS external routes	ospf	150
BGP routes	bgp	170

Trace statement

Trace statements control tracing options. GateD has global and per-protocol tracing options. Each of the sublayers of options inherit the trace option from the previous layer. For example, you can have trace options on a BGP peer statement, which would inherit any of the following trace options, if specified: BGP group, BGP statement, and the global statement, in that order.

Note: Not all of the trace options described in the following sections apply to all of the protocols. In some cases, using them does not make sense, for example, RIP does not have a state machine, and in some instances the requested tracing has not been implemented, for example such as RIP support of the **`policy`** option.

When protocols inherit their tracing options from the global tracing options, tracing levels 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, tracing options that are not explicitly specified are inherited from the global options in effect at the end of the configuration file.

Note: In a production environment, you do not want to run with tracing on. Tracing should be used for debugging purposes only. Please contact Customer Support for more information.

Traceoptions syntax

The **`traceoptions`** statement syntax is as follows:

```
traceoptions [ trace_file [ replace ] [ size size [ k|m ] files files ] ]
        [ control_options ] trace_options [ except trace_options ] ;
traceoptions none ;
```

Traceoptions options

The **`traceoptions`** statement can contain one or more of the following options:

`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.

`replace`

Tracing should start by replacing an existing file. The default is to append to an existing file.

`size` *`size`* [**`k|m`**] **`files`** *`files`*

Limits the maximum size of the trace file to the specified size (minimum 10k). When the trace file reaches the specified size, it is renamed to file.0, then file.1, file.2, up to the maximum number of files. The minimum specification is 2.

`control_options`

Specifies options that control the appearance of tracing. The valid value for this variable is `nostamp`, which specifies that a timestamp should not be prepended to all trace lines.

`trace_options`

See the following sections for more information on the global tracing options.

`except` *`trace_options`*

Used to enable a broad class of tracing and then disable more specific options.

`none`

Specifies that all tracing should be turned off for this protocol or peer.

Protocol significant trace options

You should only use the **`normal`** option flag. Other options are available, but not recommended.

`normal`

Trace normal protocol occurrences; abnormal protocol occurrences are always traced.

Packet tracing

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`

`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 is used to provide additional detail on the contents of the packet.

`send` or **`recv`**

These options limit the tracing to packets sent or received. Without these options, both sent and received packets are traced.

Note: If specified, **`detail`** must be before **`send`** or **`recv`**. If a protocol allows for several different types of packet tracing, modifiers can be applied to each individual type. But, be aware that within one tracing specification the trace flags are summed up, so specifying **`detail`** packets turns on full tracing for all packets.

Directive statement

Directive statements provide direction to the GateD configuration language parser about included files and the directories in which these files reside.

Directive statements are immediately acted upon by the parser. Other statements terminate with a semicolon (;), but directive statements terminate with a new line. The two directive statements are as follows:

`%directory` *`directory`*

Defines the directory where the `include` files are stored. When it is used, GateD looks in the directory identified by pathname for any included files that do not have a fully qualified filename. This statement does not actually change the current directory, it just specifies the prefix applied to included file names.

`%include` *`filename`*

Identifies an `include` file. The contents of the file is included in the `/etc/gated.conf` file at the point in the `/etc/gated.conf` file where the **`%include`** directive is encountered. If the filename is not fully qualified, it is considered to be relative to the directory defined in the **`%directory`** directive. The **`%include`** directive statement causes the specified file to be parsed completely before resuming with this file. Nesting up to ten levels is supported.

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

Note: These statements must begin in the first character column of the `/etc/gated.conf` file. In other words, there can be no tabs or spaces before the **`%`**. The directory or filename must be enclosed in double quotes: for example, **`%directory`** *`"/etc/include"`* or **`%include`** *`"filter.testroutes"`*. Also, comments cannot be on the same line as a **`directive`** statement.

Options statement

The **`options`** statement allows specification of some global options. If used, **`options`** must appear before any other type of configuration statement in the `/etc/gated.conf` file.

Syntax

The **`options`** statement syntax is as follows:

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

Options

The **`options`** statement can contain one or more of the following options:

`gendefault` `[` **`precedence`** *`precedence`* `] [` **`gateway`** *`gateway`* `]`

When **`gendefault`** is enabled, and when a BGP neighbor is up, it creates a default route with the special protocol **`default`**. This can be disabled per BGP group with the **`nogendefault`** option. By default, this route has a precedence of 20. This route is normally not installed in the kernel forwarding table; it is only present so it can be announced to other protocols. If a gateway is specified, the default route is installed in the kernel forwarding table with a nexthop of the listed gateway.

> **Note:** Using a more general option is preferred to using of the **`gendefault`** option. The **`gendefault`** option may not be included in future releases. See “Route aggregation and generation statements” on page 3-75 for more information on the **`generate`** statement.

`nosend`

Specifies that no packets are sent. This option 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. This is most useful for RIP, and possibly SMUX SNMP interface. This option does not yet apply to BGP and is less than useful with OSPF.

`noresolv`

By default, GateD tries 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 hosts local host and network tables. If there is insufficient routing information to send DNS queries, GateD halts during start-up. This option can be used to prevent these calls; symbolic names results in configuration file errors.

`syslog` `[` **`upto`** `]` *`log_level`*

Controls the amount of data GateD logs through **syslog** on systems where ***`setlogmask()`*** is supported. The available logging levels and other terminology are as defined in the setlogmask(3) man page. The default is equivalent to **`syslog up to`** information.

`mark` *`time`*

Specifying this option causes GateD to output a message to the trace log at the specified interval. This can be used as a method of determining if GateD is still running.

GSM statement

GateD provides an interactive interface called the GateD State Monitor (GSM) from which an operator can examine the state of route tables, interfaces, and routing protocols. In the enabled mode, the operator can also control some of the GateD operations: enabling or disabling configured traces, resetting a peering session and adjacencies, and manipulating interface tables. For more information, see “GSM command” in Chapter 4.

From the GateD configuration file (`/etc/gated.conf`), you can control access to the GSM in the following ways:

– The port on which GateD responds to the GSM connections

– The hosts from which GateD can accept the GSM connections

– The users whose passwords provide admission to the GSM

Additionally, the GSM can be turned on or off through the `/etc/gated.conf` file.

Note: For backward compatibility, GSM is on by default.

Syntax

The **`gsm`** statement syntax is as follows:

```
gsm  ( yes  |  no  |  on  |  off )
[ {
      [ port port-number ; ]
      [ usernames user-list ; ]
      [ hosts  host-set ; ]
}  ]  ;
```

Options

The **`gsm`** statement can contain one or more of the following options:

`port` *`port-number`*

Controls the TCP port to which GSM binds. By default, GSM binds to the port named **`gii`** in the `/etc/services` file. If no such port is defined, then the final default is port number 616. In other words, if this option specifies a GSM port, then that port is the one that is used by GSM. If no port is specified with the **`port`** option, and if `/etc/services` specifies a port number for a service named **`gii`**, then GateD uses that port for GSM. If neither of the two previous conditions are true, then GateD defaults to using TCP port 616 for GSM.

`usernames` *`user-list`*

Lists the users permitted to connect to the GSM; the list is in the format of double-quoted strings (for example, `“netstar”`). Multiple users are separated by white space (for example, `“netstar” “jsmith” “manderson”`). When you open a telnet session to the GSM, the GSM prompts for a user name and a password. You can continue with the GSM session if you provide a user name from the user-list and the corresponding system password. The default is to permit only the user “`netstar”`. For backward compatibility, the default case is to prompt for password only.

`hosts` *`host-set`*

Lists the hosts from which GateD that can permit telnet connections to the GSM. The syntax for the *`host-set`* is the same as for the prefix matching in the BGP **`allow`** keyword, which is as follows:

```
network
network mask mask
network ( masklen | / ) number
all
host host
```

Changes to the **`port, usernames`**, and/or **`hosts`** options requires that you execute a **`gdc reconfig`** for the changes to take effect. Also, changes to these options do not affect already-established GSM session regardless of whether you execute the **`gdc reconfig`** command.

For backward compatibility, the default case is to accept connections from any source addresses.

For example, the following is the current default GSM configuration made explicit in the `/etc/gated.conf` file:

```
gsm on {
    port        616 ;
    usernames   "netstar" ;
    hosts       all ;
} ;
```

Interface statement

An interface is the connection between a router and one of its attached networks. A physical interface can be specified by interface name, by IP address, or by domain name unless the network is an un-numbered point-to-point network.

Multiple levels of reference in the configuration language enables identification of interfaces using wildcard, interface type name, or delete word address. Use caution when using interface names as future UNIX operating systems may allow more than one address per interface.

The *`interface_list`* is a list of one or more interface names including wildcard names and names that can specify more than one interface or address, or the **`all`** keyword for all interfaces see “Interface lists” on page 3-14 for more information.

Syntax

The **`interfaces`** syntax is as follows:

```
interfaces {
      options
           [ strictinterfaces ]
           [ scaninterval time_seconds ]
           ;
      interface interface_list
           [ precedence precedence ]
           [ down precedence precedence ]
           [ passive ]
           [ simplex ]
           [ reject ]
           [ blackhole ]
           ;
      define address
           [ broadcast address ] | [ pointtopoint address ]
           [ netmask mask ]
           [ multicast ]
           ;
} ;
```

Options

The **`interfaces`** statement can contain one or more of the following options:

`options`
Allows configuration of some global options related to interfaces. The keywords are as follows:

`strictinterfaces`

Indicates that it is a fatal error to reference an interface in the configuration file that is not present when GateD is started and not listed in a **`define`** statement. Without this option, a warning message is issued but GateD continues.

`scaninterval` *`time_seconds`*

Specifies how often GateD scans the kernel interface list for changes. The default is every 15 seconds on most systems, and 60 seconds on systems that pass interface status changes through the routing socket, for example, BSD 4.4. Note that GateD also scans the interface list after receiving a `SIGUSR2`.

`interface` *`interface_list`*

Sets interface options on the specified interfaces. An interface list is **`all`** or a list of interface names (see interface warning described previously in **`options`**), domain names, or numeric addresses. The keywords are as follows:

`precedence` *`precedence`*

Sets the precedence for directly-connected routes to this interface when it is up and appears to be functioning properly. The default precedence is 0.

`down precedence` *`precedence`*

Sets the precedence for directly-connected routes to this interface when GateD does not believe it to be functioning properly, but the kernel does not indicate it is down. The default value is 120.

`passive`

Prevents GateD from changing the precedence of the route to this interface if it is not believed to be functioning properly due to lack of received routing information. GateD only performs this check if the interface is actively participating in a routing protocol.

`simplex`

Defines an interface as unable to hear its own broadcast 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, packets from myself are assumed to have been looped back in software, and are not used as an indication that the interface is functioning properly.

`reject`

Specifies that the address of the interface that matches these criteria are used as the local address when installing **`reject`** routes in the kernel. Should only be used with systems based on BSD 4.3 Tahoe or earlier that have installed a **`reject/blackhole`** pseudo interface.

`blackhole`

Specifies that the address of the interface that matches these criteria are used as the local address when installing **`reject`** routes in the kernel. Should only be used with systems based on BSD 4.3 Tahoe or earlier that have installed a **`reject/blackhole`** pseudo interface.

`define` *`address`*

Defines interfaces that might not be present when GateD is started so they can be referenced in the configuration file when **`strictinterfaces`** is defined. The keywords are as follows:

`broadcast` *`address`*

Defines the interface as broadcast capable, for example, Ethernet or Token Ring, and specifies the broadcast address.

`pointtopoint` *`address`*

Defines the interface as a point-to-point interface, for example, SLIP or PPP, and specifies the address on the local side. The first *`address`* on the **`define`** statement references the address of the host on the **`remote`** end of the interface, the *`address`* specified after this **`pointtopoint`** keyword defines the address on the local side of the interface.

An interface not defined as **`broadcast`** or **`pointtopoint`** is assumed to be non-broadcast multi-access (NBMA), such as an X.25 network.

`netmask` *`mask`*

Specifies the subnet mask to be used on this interface. This is ignored on **`pointtopoint`** interfaces.

`multicast`

Specifies that the interface is capable of multicast.

Interface lists

The *`interface_list`* is a list of references to interfaces or groups of interfaces. There are four methods available for referring to interfaces; they are listed as follows from most general to most specific:

`all`

Refers to all available interfaces.

`interface name wildcard`

Refers to all the interfaces of the same type. UNIX interfaces consist of the name of the device driver, for example, `ie`, and a unit number, for example, 0, 5, 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 refers to all Interlan Ethernet interfaces, while `le` refers to all Lance Ethernet interfaces. In this example, `ie` would not match `ie10`.

`Interface name`

Refers to a specific interface, usually one physical interface. These are specified as an alphabetic part followed by a numeric part that matches one specific interface. Be aware that on many systems, there can be more than one protocol address on a specific physical interface. For example, `ef1` matches an interface named `ef1`, but not an interface named `ef10`.

`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.dnn.mil`. Note that a symbolic hostname reference is only valid when it resolves to only one address. Lucent does not recommend using symbolic hostnames.

If many interface lists are present in the configuration file with more than one parameter, 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.

For example, consider the following system with three interfaces, `le0, le1`, and `du0`:

```
rip yes {
    interface all noripin noripout ;
    interface le0 ripin ;
```

```
        interface le1 ripout ;
    } ;
```

RIP packets are accepted from interfaces `le0` and `le1`, but not from `du0`. RIP packets would only be sent on interface `le1`.

IP interface addresses and routes

The BSD 4.3 and later networking implementations allow the following four types of interfaces:

`loopback`

This interface must have the address of 127.0.0.1. Packets sent to this interface 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. It is useful to assign an additional address to this interface that is the same as the OSPF or BGP router ID. This allows routing to a system based on the router ID that works if some interfaces are down.

`broadcast`

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

`point-to-point`

This is a tunnel to another host, usually on a serial link. This interface has a local address, and a remote address. Although it can 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 can be shared among many point-to-point and up to one non-point-to-point interface. This is technically a form of the router id method for address-less links. This technique conserves subnets as none are required when using this technique.

If a subnet mask is specified on a point-to-point interface, it is only used by RIP version 1 to determine which subnets can be propagated to the router on the other side of this interface.

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

This type of interface is multi-access, but not capable of broadcast. Examples are Frame Relay and X.25. This type of interface has a local address and a subnet mask.

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; GateD does it to ensure consistency.

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 precedence 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 are returned by the router on the remote end. This is used to verify operation of the link. Because OSPF installs routes with a precedence of 10, these routes override the route installed with a precedence 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 interface with a precedence of 0 that is not 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, that then take the appropriate action. GateD assumes that interfaces 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 also ignores any point-to-point interface that has the same local and remote addresses, it assumes it is in a loopback test mode.

Definition statement

Definition statements are general configuration statements that relate to all of GateD or at least to more than one protocol. The five definition statements are **`autonomoussystem`**, **`confederation`**, **`routerid`**, **`routing-domain`**, and **`martians`**. When used, these statements must appear before any other type of configuration statement in the `/etc/gated.conf` file.

Also, you must use **`confederation`** and **`routing-domain`** together, but you cannot use **`confederation`**, **`routing-domain`**, and **`autonomoussystem`** together. They are mutually exclusive for configuring BGP.

Syntax

The **`definition`** statements syntax is as follows:

```
autonomoussystem autonomous_system [ loops number ] ;
multipath (  yes  |  no  |  off  |  no )  ;
routerid host ;
confederation confederation ;
routing-domain rdi ;
martians {
          host host [ allow ] ;
          network [ allow ] ;
          network mask mask [ allow ] ;
          network ( masklen | / ) number [ allow ] ;
          default [ allow ] ;
} ;
```

Options

The definitions statements contain one or more of the following:

`autonomoussystem` *`autonomous_system`* `[` **`loops`** *`number`* `]`

Sets the autonomous system number of the router to *`autonomous_system`*. This option is required if BGP is used. The AS number is assigned by the Network Information Center (NIC).

The **`loops`** keyword is only for protocols supporting AS paths, such as BGP. It controls the number of times this autonomous system can appear in an AS path and defaults to 1 (one).

`multipath` `(` **`yes`** `|` **`no`** `|` **`off`** `|` **`on`** `)`

The Equal Cost Multipath Configuration (ECMP) **`multipath`** parameter enables/disables GateD's ability to install multiple gateways for network or host prefixes in the kernel route table. The default is **`off`**.

The **`on`** option is the same as the **`yes`** option, and enables GateD to install multiple routes for a single source with the same destination but different nexthops. The **`off`** option is the same as the **`no`** option, and means that each route GateD installs in the kernel has a unique destination and nexthop. ECMP supports prefixes learned through the OSPF and OSPF_ASE protocols. BGP is supported if OSPF is used to resolve BGP nexthops.

The **`multipath`** keyword cannot be changed through the **`gdc reconfig`** command. Changes take effect only when GateD is started (**`gdc start`**) or restarted (**`gdc restart`**).

`routerid` *`host`*

Sets the router identifier for use by the BGP and OSPF protocols. The default is the address of the first interface encountered by GateD. The address of a non-point-to-point interface is preferred over the local address of a point-to-point interface. An address on a loopback interface that is not the loopback address (127.0.0.1) is preferred.

`confederation` *`confederation`*

Sets the confederation identifier for use by the BGP protocol for the configuration of `group type confed`. See the “BGP statement” on page 3-28 for more information on configuring confederations. Note that **`autonomoussystem`** and **`confederation`** are mutually exclusive; they cannot both be defined within the same `/etc/gated.conf`. Also, note that **`confederation`** and **`routing-domain`** both must be set within the same `/etc/gated.conf`.

`routing-domain` *`rdi`*

Sets the routing domain identifier for use by the BGP protocol for the configuration of group type confed. See the “BGP statement” on page 3-28 for more information on configuring confederations. Note that **`confederation`** and **`routing-domain`** both must be set within the same `/etc/gated.conf`.

`martians`

Defines a list of addresses about which all routing information is ignored.

Sometimes a misconfigured system sends out obviously invalid destination addresses. These invalid addresses, called **`martians`**, are rejected by the routing software. This command allows additions to the list of **`martian`** addresses. See “Route filters” on page 3-65 for more information on specifying ranges.

Also, the **`allow`** parameter can be specified to explicitly allow a subset of a range that is disallowed.

Example

The following example shows how the definition statements can be used. The **`options`** statement tells the system to generate a default route when it peers with a BGP neighbor. The **`autonomoussytem`** statement tells GateD to use AS number 249 when participating in BGP. The **`interface`** statement tells GateD not to mark interface 128.66.12.2 as down even if it sees no traffic.The **`martians`** statement prevents the host route 10.10.10.26 and the network 10.8.0.0/16 from being installed:

```
options gendefault ;

autonomoussytem 249 ;

interface 128.66.12.2 passive ;

martians {

host 10.10.10.26 ;

10.8.0 mask 255.255.0.0 ;

};
```

RIP statement

The **rip** statement enables or disables RIP. If the **rip** statement is not specified, the default is **rip off ;**. If enabled, RIP assumes **nobroadcast** when there is only one interface and **broadcast** when there is more than one.

Syntax

The **rip** statement syntax is as follows:

```
rip  ( yes | no | on | off  ) [ {
      broadcast ;
      nobroadcast ;
      nocheckzero ;
      precedence precedence ;
      defaultmetric metric ;
      query authentication [none |[ simple | md5 ] password)] ;
      interface interface_list
            [ noripin ] | [ ripin ]
            [ noripout ] | [ ripout ]
            [ metricin metric ]
            [ metricout metric ]
               [ version 1 ]|[ version 2 [ multicast|broadcast ]]
             [ [ secondary ] authentication [ none |
            ( [ simple | md5 ] password ) ] ] ;
      trustedgateways gateway_list ;
      sourcegateways gateway_list ;
      traceoptions trace_options ;
} ] ;
```

Options

The **rip** statement can contain one or more of the following options:

broadcast

Specifies that RIP packets are broadcast regardless of the number of interfaces present. This is useful when propagating static routes or routes learned from another protocol into RIP. In some cases, the use of **broadcast** when only one network interface is present can cause data packets to traverse a single network twice.

nobroadcast

Specifies that RIP packets are not broadcast on attached interfaces, even if there are more than one. If a **sourcegateways** clause is present, route advertisements are still unicast directly to the specified gateway.

`nocheckzero`

Specifies that RIP should not make sure that reserved fields in incoming RIP version 1 packets are zero. Normally RIP rejects packets where the reserved fields are zero.

`precedence` *`precedence`*

Sets the precedence for routes learned from RIP. The default precedence is 100. This precedence can be overridden by a precedence specified in the import policy. This keyword can also be stated as **`protocol-precedence`** or **`proto-precedence`**.

`defaultmetric` *`metric`*

Defines the metric used when advertising routes through RIP that were learned from other protocols. If not specified, the default value is 16 (unreachable). This choice of values requires you to explicitly specify a metric in order to export routes from other protocols into RIP. This metric can be overridden by a metric specified in export policy.

`query authentication` `[` **`none`** `| ( [` **`simple`** `|` **`md5`** `]` *`password`* `) ]`

Specifies the authentication required of query packets that do not originate from routers. The default is **`none`**.

`interface` *`interface_list`*

Controls various attributes of sending RIP on specific interfaces. See “Interface lists” on page 3-14“ for more information.

If there are multiple interfaces configured on the same subnet, RIP updates are only sent from the secondary interfaces if they are declared on the Interface statement by IP number (not by logical interface name). Also, IP aliases for the `lo0` (loopback) interface must also be declared by IP number. Under no circumstances is the export or designation of `lo0` or 127.0.0.1 allowed.

Parameters

The parameters are as follows:

`noripin`

Specifies that RIP packets received through the specified interface is ignored. The default is to listen to RIP packets on all non-loopback interfaces.

`ripin`

This is the default. This argument can be necessary when **`noripin`** is used on a wildcard interface descriptor.

`noripout`

Specifies that no RIP packets are sent on the specified interfaces. The default is to send RIP on all broadcast and non-broadcast interfaces when in **`broadcast`** mode. The sending of RIP on point-to-point interfaces must be manually configured.

`ripout`

This is the default. This argument is necessary when it is desired to send RIP on point-to-point interfaces and can be necessary when **`noripin`** is used on a wildcard interface descriptor.

`metricin` *`metric`*

Specifies the RIP metric to add to incoming routes before they are installed in the routing table. The default is the kernel interface metric plus 1which is the default RIP hop count. If this value is specified, it is used as the absolute value, the kernel metric is not added. This option is used to make this router prefer RIP routes learned through the specified interface(s) less than RIP routes from other interfaces.

metricout *metric*

Specifies the RIP metric to be added to routes that are sent through the specified interface(s). The default is zero. This option is used to make other routers prefer other sources of RIP routes over this router.

version 1

Specifies that the RIP packets sent on the specified interface(s) is RIPv1 packets. This is the default.

version 2

Specifies that RIPv2 packets are sent on the specified interfaces(s). If IP multicast support is available on the specified interface(s), the default is to send full RIPv2 packets. If multicast support is not available, RIPv1-compatible RIPv2 packets are sent.

multicast

Specifies that RIPv2 packets should be multicast on this interface. This is the default.

broadcast

Specifies that RIPv1-compatible RIPv2 packets should be broadcast on this interface, even if IP multicast is available.

[**secondary**] **authentication** [**none** | ([**simple** | **md5**] *password*)]

This defines the authentication type to use. It applies only to RIPv2 and is ignored for RIPv1 packets. The default authentication type is **none**. If a password is specified, the authentication type defaults to **simple**. The password should be a quoted string with between 0 and 16 characters.

If **secondary** is specified, this defines the secondary authentication. If omitted, the primary authentication is specified. The default is primary authentication of none and no secondary authentication.

trustedgateways *gateway_list*

Defines the list of gateways from which RIP accepts updates. The *gateway_list* is simply a list of host names or IP addresses. By default, all routers on the shared network are trusted to supply routing information. But, if the **trustedgateways** clause is specified, only updates from the gateways in the list are accepted.

sourcegateways *gateway_list*

Defines a list of routers to which RIP sends packets directly, not through multicast or broadcast. This can be used to send different routing information to specific gateways. Updates to gateways in this list are not affected by **noripout** on the interface.

traceoptions *trace_options*

Specifies the tracing options for RIP. See the RIP-specific tracing options in the following section for more information.

Tracing options

The policy option logs information whenever a new route is announced, or the metric being announced changes, or a route enters or leaves holddown.

Packet tracing options that can be modified with **detail**, **send** or **recv** are as follows:

packets

All RIP packets.

request

RIP information request packets, such as REQUEST, POLL and POLLENTRY.

response

RIP RESPONSE packets, that are the type of packet that actually contains routing information.

other

Any other type of packet. The only valid ones are TRACE_ON and TRACE_OFF, both of which are ignored.

OSPF statement

The OSPF statement enables you to configure GateD for the OSPF protocol.

Syntax

The **ospf** statement syntax is as follows:

```
ospf  ( yes | no | on | off  ) [ {
        defaults {
               precedence precedence ;
               cost cost ;
               tag [ as ] tag ;
               type 1 | 2 ;
               inherit-metric ;
      } ;
       exportlimit routes ;
       exportinterval time_seconds ;
       traceoptions trace_options ;
       monitorauthkey authkey ;
       monitorauth none | ( [ simple | md5 ] authkey ) ;
       backbone | ( area area ) {
             authtype  none | simple ;
             stub [ cost cost ] ;
            networks {
                 network [ restrict ] ;
                 network mask mask [ restrict ] ;
                 network ( masklen | / ) number [ restrict ] ;
                 host host [ restrict ] ;
            } ;
            stubhosts {
                  host  cost cost ;
            } ;
            interface interface_list ; [ cost cost ] {
                   interface_parameters
```

```
                } ;
                 interface interface_list nonbroadcast [ cost cost ] {
                      pollinterval time_seconds ;
                      routers {
                            gateway [ eligible ] ;
                      } ;
                      interface_parameters
                } ;
                # Backbone only:
                  virtuallink neighborid router_id  transitarea area {
                       interface_parameters
                 } ;
            } ;
       } ] ;
```

Note: The following are the *interface_parameters* referred to in the **ospf** syntax statement. These can be specified on any class of interface and are described under the **interface** clause.

```
       enable | disable | passive ;
    retransmitinterval time_seconds ;
    transitdelay time_seconds ;
    priority priority ;
    hellointerval time_seconds ;
    routerdeadinterval time_seconds ;
    authkey auth_key ;
```

Options

defaults

These options specify the defaults used when importing OSPF ASE routes into the GateD routing table and exporting routes from the GateD routing table into OSPF ASEs.

precedence *precedence*

Sets the global precedence for OSPF ASE incoming routes. This precedence can be overridden by a precedence specified by the import policy. The default precedence for OSPF ASE routes is 150. This keyword can also be stated as **protocol-precedence** or **proto-precedence**.

cost *cost*

The cost is used when exporting a non-OSPF route from the GateD routing table into OSPF as an ASE. This can be explicitly overridden in export policy by the **metric** keyword. The default value is 1.

Note: This parameter has no effect on the **cost** keyword on the interface command.

`tag` `[` **`as`** `]` *`tag`*

OSPF ASE routes have a 32-bit tag field that is not used by the OSPF protocol, but can be used by export policy to filter routes. When OSPF is interacting with an EGP, 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 zero.

`type` *`1 | 2`*

Routes imported from BGP into OSPF default to becoming type 2 ASEs. This default can be explicitly changed here and overridden in export policy. For more information on type, see the “Introduction to OPSF” section in Chapter 1.

`inherit-metric`

Inherit-metric allows an OSPF ASE route to inherit the metric of the external route from the BGP MED when no metric is specified on the export. If this value is specified in more than one place, the following rules of precedence apply in the order shown:

– A metric specified on the **`export`** statement.

– Inherit-metric is specified in the **`defaults`** statement.

– Cost is specified in the **`defaults`** statement.

– The default value is 1.

`exportlimit` *`routes`*

Specifies how many ASEs are generated and flooded in each batch. The default is 100.

`exportinterval` *`time_seconds`*

Specifies how often a batch of ASE link state advertisements is generated and flooded into OSPF. Time is specified in seconds. The default is once per second.

`traceoptions` *`trace_options`*

Specifies the tracing options for OSPF. See “Trace statement” on page 3-6 and the OSPF specific tracing options in the following subsection.

`monitorauthkey` *`authkey`*

OSPF state can be queried using the `ospf_monitor` utility. This utility sends non-standard OSPF packets that generate a text response from OSPF. By default, these requests are not authenticated. If an authentication key is configured, the incoming requests must match the specified authentication key. No OSPF state can be changed by these packets, but the act of querying OSPF can utilize system resources.

`backbone area` *`area`*

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 can only be configured using the **`backbone`** keyword, it cannot be specified as area 0. The backbone interface can be a **`virtuallink`**.

`authtype none | simple`

OSPF specifies an authentication scheme per area. Each interface in the area must use this same authentication scheme, although it can use a different **`authenticationkey`**. The currently valid values are **`none`** for no authentication, or **`simple`** for simple password authentication. The default is **`none`**.

If **`authkey simple`** is specified but there is no **`authkey`** *`xxx`* in the `/etc/gated.conf` file, this is equivalent to specifying **`authkey none`**. In both cases, authentication is disabled.

`stub` `[` **`cost`** *`cost`*`]`

A **`stub`** area is one in which there are no ASE routes. Specifying **`stub`** prevents participating routers from advertising ASE routes into that area. If a **`cost`** is specified, this is used to inject a default route into the area with the specified ***`cost`***. This is not valid for backbone area. The **`stub`** keyword must be specified on all routers within a given area.

`networks`

The **`networks`** list describes the scope of an area. 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 are not advertised. Intra-area LSAs that do not fall into any range are also advertised as summary network LSAs. This option is very useful on well-designed networks in reducing the amount of routing information propagated between areas. The entries in this list are either networks, or a subnetwork/mask pair. See “Route filtering” on page 3-53 for more detail about specifying ranges.

`stubhosts`

This list specifies directly-attached hosts that should be advertised as reachable from this router and the costs they should be advertised with. 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 stubhosts. 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. This is more reliable using an interface address as a router ID, because that address is unavailable if the interface goes down.

`interface` *`interface_list`* `[` **`cost`** *`cost`* `]`

This form of the interface clause is used to configure a broadcast, that requires IP multicast support, or a point-to-point interface. See “Interface lists” on page 3-14 for the description of the *`interface_list`*.

Each interface has a *`cost`*. The costs of all interfaces a packet must cross to reach a destination are summed to get the cost to that destination. The default cost is one, but another non-zero value can be specified.

Interface parameters common to all types of interfaces are as follows:

`enable | disable | passive`

Enable or disable OSPF on the interface. If **`passive`** is used, OSPF behaves as though it is enabled on the interface, the interface's routes are included in LSAs issued by the router, but OSPF packets are not sent or accepted on the interface. This can be useful for advertising DMZs or other interconnect networks into OSPF, as an alternative to advertising them into ospfase.

`retransmitinterval` *`time_seconds`*

The number of seconds between link state advertisement retransmissions for adjacencies belonging to this interface.

`transitdelay` *`time_seconds`*

The estimated number of seconds required to transmit a link state update over this interface. **`transitdelay`** takes into account transmission and propagation delays and must be greater than 0.

`priority` *`priority`*

A number between 0 and 255 specifying the priority for becoming the designated router on this interface. When two routers attached to a network both attempt to become the designated router, the one with the highest priority wins. A router whose router priority is set to 0 is ineligible to become designated router.

OSPF supports both NBMA and P2P interfaces. The priority for these interfaces must be manually configured to elect the designated router.

If no priority keyword is present, priority defaults to 0. At least one router on a network must have a non-zero priority if OSPF is to work at all.

`hellointerval` *`time_seconds`*

The length of time, in seconds, between Hello packets that the router sends on the interface.

`routerdeadinterval` *`time_seconds`*

The number of seconds not hearing a router's Hello packets before the router's neighbors declare it down. All routers on the same network must use the same value for **`routerdeadinterval`**.

`authkey` *`auth_key`*

Used by OSPF authentication to generate and verify the authentication field in the OSPF header. The authentication key can be configured on a per-interface basis. It is specified by one to eight decimal digits separated by periods, or a one- to eight- character string in double quotes. The password “01.02.03” is equivalent to “1.2.3” because the fields are treated as numeric values, not just as an ASCII string. Similarly, the password “0” is equivalent to “0.0.0.0”.

If no **`authkey`** keyword is present, authentication is disabled exactly as if **`authtype none`** has been specified. Specifying an **`authkey`** on an interface without also specifying **`authtype`** on the area results in an arcane parser error.

Point-to-point interfaces also support the following additional parameter:

`nomulticast`

By default, OSPF packets to neighbors on point-to-point interfaces are sent through the IP multicast mechanism. Some implementations of IP multicasting for UNIX have an error that precludes the use of IP multicasting on these interfaces. GateD detects this condition and falls back to sending unicast OSPF packets to this point-to-point neighbor.

If the use of IP multicasting is not desired because the remote neighbor does not support it, the **`nomulticast`** parameter can be specified to force the use of unicast OSPF packets. This option can also be used to eliminate warnings when GateD detects the error mentioned above.

This option can only be used on interfaces configured with a point-to-point flag using `ifconfig`.

`interface` *`interface_list`* **`nonbroadcast`** `[` **`cost`** *`cost`* `]`

This form of the interface clause is used to specify a non-broadcast multi-access (NBMA) interface. Because an OSPF **`broadcast`** medium must support IP multicasting, a broadcast-capable medium, such as Ethernet, that does not support IP multicasting must be configured as a non-broadcast interface.

A non-broadcast interface supports any of the standard **`interface`** clauses listed above, plus the following two that are specific to non-broadcast interfaces:

`pollinterval` *`time`*

Before adjacency is established with a neighbor, OSPF packets are sent periodically at the specified **`pollinterval`**.

`routers {` *`ipaddr ; ... ipaddr ;`* **`} ;`**

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

`virtuallink neighborid` *`router_id`* **`transitarea`** *`area`*

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 transit *`area`* specified must also be configured on this system. All standard interface parameters defined by the **`interface`** clause above can be specified on a virtual link.

Tracing options

In addition to the following OSPF specific trace flags, OSPF supports the **`state`** that traces interface and neighbor state machine transitions.

`lsabuild`

Link State Advertisement creation.

`spf`

Shortest Path First (SPF) calculations.

Packet tracing options that can be modified with **`detail`**, **`send`** and **`recv`** are as follows:

`hello`

OSPF Hello packets that are used to determine neighbor reachability.

`dd`

OSPF Database Description packets that are used in synchronizing OSPF databases.

`request`

OSPF Link State Request packets that are used in synchronizing OSPF databases.

`lsu`

OSPF Link State Update packets that are used in synchronizing OSPF databases.

`ack`

OSPF Link State Ack packets that are used in synchronizing OSPF databases.

BGP statement

The **bgp** statement enables or disables BGP. By default, BGP is disabled. The default metric for announcing routes through BGP is no metric.

Syntax

The **bgp** statement syntax is as follows:

```
bgp  ( yes | no | on | off )
{
     [ precedence precedence ; ]
     [ preference preference ; ]
     [ allow bad community ; ]
     [ defaultmetric metric ; ]
     [ traceoptions trace_options ; ]
     [ clusterid host ; ]
     [ disable export best ; ]
[ group type
     ( external peeras autonomous_system
          [ ignorefirstashop ]  [ subgroup integer ]
          [ metricout metric ] [ med ]  [ nexthopself ]
          [ send-no-community | send-community ]
          { bgp_peer_stmts } ;

   | ( internal peeras autonomous_system
           [ reflector-client [ no-client-reflect ] ]
           [ ignorefirstashop ]
           [ lcladdr local_address ]
           [ outdelay time ]
           [ metricout metric ]
           [ subgroup integer ]
           [ nexthopself ]
           [ send-no-community | send-community ]
           { bgp_peer_stmts } ;

   | ( routing peeras autonomous_system proto proto_list
             interface interface_list
           [ reflector-client [ no-client-reflect ] ]
           [ ignorefirstashop ]
           [ lcladdr local_address ]
```

```
                [ outdelay time ]
                [ metricout metric ]
                [ subgroup integer ]
                [ nexthopself ]
                [ send-no-community | send-community ]
                { bgp_peer_stmts } ;

        | ( confed peeras autonomous_system proto proto_list
                  interface interface_list
                [ reflector-client [ no-client-reflect ] ]
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ reflector-client [ no-client-reflect ] ]
                [ subgroup integer ]
                [ nexthopself ]
                [ send-no-community | send-community ]
                { bgp_peer_stmts } ;

        | ( test peeras autonomous_system ) ]
            {
                [ allow {
                  network
                  network mask mask
                  network ( masklen | / ) number
                  all
                  host host ]
             } ;
                  peer host
                [ metricout metric ]
                [ ignorefirstashop ]
                [ nogendefault ]
                [ gateway gateway ]
                [ precedence precedence ]
                [ preference preference ]
                [ lcladdr local_address ]
                [ holdtime time ]
                [ version number ]
```

```
                    [ passive ]
                    [ sendbuffer number ]
                    [ recvbuffer number ]
                    [ outdelay time ]
                    [ keep [ all | none ] ]
                    [ show-warnings ]
                    [ noaggregatorid ]
                    [ keepalivesalways ]
                    [ v3asloopokay ]
                    [ nov4asloop ]
                    [ ascount count ]
                    [ throttle count ]
                    [ allow bad routerid ]
                    [ logupdown ]
                    [ ttl ttl ]
                    [ traceoptions trace_options ]
                    [ nexthopself ]
                    ;
                } ;
            } ;
```

Options

The **bgp** statement can contain one or more of the following options:

precedence *precedence*

Sets the global precedence for BGP incoming routes. The default precedence is 170. This precedence can be overridden by a precedence specified on the **peer** statement, or by the import policy. This keyword can also be stated as **protocol-precedence** or **proto-precedence**.

preference *preference*

Sets the preference of all BGP peers in the group types' see "Group and peer parameters" on page 3-31 for more information. This preference can be used to prefer routes learned from one group of peers over others. This preference can be overridden by preference set for a particular peer or by the import policy. This keyword can also be stated as **pref**.

`allow bad community`

BGP communities are specified in RFC 1997 as being a 16-bit AS number followed by an arbitrary 16-bit integer; these are referred to these as the "AS part" and the "tag part", respectively. The specification explicitly reserves communities with AS part 0 or 65535 for use as well-known communities or other future uses.

Accordingly, GateD ordinarily does not permit use of 0 or 65535 in the AS part of a community. However, some BGP implementations have been found to allow this behavior. In order to permit interoperation with such implementations, **`allow bad community`** can be used within the BGP clause. The preferred solution is to configure the other routers in use to conform to RFC 1997 by using a valid AS number, normally, an AS number assigned to you by the InterNIC, in the AS part.

On a router that configures communities as 32-bit integers rather than as an AS part and a tag part, the reserved communities are 0 through 65535 and 4294901760 through 4294967295. Avoid using these communities. Any other community (65536 through 4294901759) is legal, although it is advisable to use one's own AS number in the AS part, for example, communities 80871424 through 80936959 have AS 1234 in the AS part.

`defaultmetric` *`metric`*

Defines the metric used when advertising routes through BGP. If not specified, no metric is propagated. This metric can be overridden by a metric specified on the neighbor or group statements or in export policy.

`traceoptions` *`trace_options`*

Specifies the tracing options for BGP. By default, these are inherited from the global trace options. These values can be overridden on a group or neighbor basis. See “Trace statement” on page 3-6 and the BGP-specific tracing options for more information.)

`clusterid` *`host`*

Specifies the route reflection cluster ID for BGP. The cluster ID defaults to be the same as the 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 is that IDs of clusters within an AS must be unique within that AS, and 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 always fulfills these criteria. If there is only one route reflector in the cluster, the **`clusterid`** setting can be omitted, as the default is sufficient.

`disable export best`

Disables the configurable export-best BGP (CEBB) functionality, which is on by default. CEBB allows GateD to export BGP routes that are not actually installed because there is an IGP route that has taken precedence over the BGP route. See “Additions to BGP” in Chapter 1 for more information on CEBB.

Group and peer parameters

The BGP statement has **`group`** clauses and **`peer`** subclauses. A **`group`** clause usually defines default parameters for a group of peers. These parameters apply to all subsidiary **`peer`** subclauses. Any number of groups can be specified, but each must have a unique combination of **`type`**, **`peers`**, and **`nexthopself`** options.

BGP peers are grouped by type and the autonomous system of the peers. Any number of **`peer`** subclauses can be specified within a group. Some of the parameters from the **`peer`** subclause can be specified on the **`group`** clause to provide defaults for the whole group which can be overridden for individual peers.

The general rule for using the **`group`** and **`peer`** options is that any option can appear in either a **`group`** or **`peer`** statement. Options in the group statement apply to all of the peering sessions for that group. Options in a **`peer`** subclause have no effect on the other peering sessions. The following constraints apply:

- Do not specify both the **`gateway`** and **`lcladdr`** options.
- Do not specify a **`subgroup`** on a peer; this can only be done on a group basis.
- Do not specify a **`holdtime`** of less than six seconds.

For groups of **`internal`**, **`routing`**, and **`confed`** types, the following constraints apply:

- Do not specify **`metricout`** on the peer statement.
- Do not specify **`outdelay`** on the peer statement.
- Do not specify **`nexthopself`** on the peer statement.
- Do not specify **`ascount`** on these group types; it applies only to peers in external groups.

Group types

The following **`group`** types are allowed:

External

```
group type external peeras autonomous_system [ med ]
[ ignorefirstashop ] [ subgroup integer]  [ nexthopself ]
[ send-no-community | send-community ] { bgp_peer_stmts } ;
```

In the classic external BGP group, full policy checking is applied to all incoming and outgoing advertisements. The external neighbors must be directly reachable through one of the machine's local interfaces. The nexthop transmitted is computed with respect to the shared interface.

If the **`gateway`** parameter is present on peer statement, EBGP peers can be on different networks. Refer to the **`gateway`** keyword statement described later in this BGP section.

Routing - an internal group that uses the routes of an interior protocol to resolve forwarding addresses.

```
group type routing peeras autonomous_system proto proto
[ interface interface_list ] [ reflector-client [no-client-reflect ] ]
[ ignorefirstashop] [ lcladdr local_address ] [ outdelay time ]
[ metricout metric ] [ subgroup integer ]  [ nexthopself ]
[ send-no-community | send-communitay ] { bgp_peer_stmts } ;
```

A **`type routing`** group propagates external routes between routers that are not directly connected, and computes immediate nexthops for these routes by using the BGP nexthop that arrived with the route as a forwarding address to be resolved through an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate nexthops for the former.

Confed - An internal group that uses the routes of an interior protocol to resolve forwarding addresses.

> **`group type confed peeras`** *`autonomous_system`* **`proto`** *`proto_list`*
>
> **`interface`** *`interface_list`* `[` **`reflector-client`** `[` **`no-client-reflect`** `] ]`
>
> `[` **`ignorefirstashop`** `] [` **`lcladdr`** *`local_address`* `] [` **`outdelay`** *`time`* `]`
>
> `[` **`metricout`** *`metric`* `] [` **`subgroup`** *`integer`* `] [` **`nexthopself`** `]`
>
> `[` **`send-no-community`** `|` **`send-community`** `] { bgp_peer_stmts };`
>
> A **`type confed`** group propagates external routes between routers that are not directly connected, and computes immediate nexthops for these routes by using the BGP nexthop that arrived with the route as a forwarding address to be resolved through an internal protocol's routing information. In essence, internal BGP is used to carry AS external routes, while the IGP is expected to only carry AS internal routes, and the latter is used to find immediate nexthops for the former.

Internal - an internal group operating where there is no IP-level IGP, for example, an SMDS network or MILNET.

> **`group type internal peeras`** *`autonomous_system`* `[` **`ignorefirstashop`** `]`
>
> `[` **`reflector-client`** `[` **`no-client-reflect`** `] ] [` **`metricout`** *`metric`* `]`
>
> `[` **`lcladdr`** *`local_address`* `] [` **`outdelay`** *`time`* `] [` **`subgroup`** *`integer`* `]`
>
> `[` **`nexthopself`** `] [` **`send-no-community`** `|` **`send-community`** `]`
> `{ bgp_peer_stmts };`
>
> All neighbors in this group are required to be directly reachable through a single interface. All next-hop information is computed with respect to this interface. Import and export policy can be applied to group advertisements. Routes received from external BGP neighbors are by default readvertised with the received metric.

Test - an extension to external BGP that implements a fixed policy using test peers.

> **`group type test peeras`** *`autonomous_system`*
>
> Fixed policy and special case code make test peers relatively inexpensive to maintain. Test peers do not need to be on a directly-attached network. If GateD and the peer are on the same (directly-attached) subnet, the advertised nexthop is computed with respect to that network. Otherwise, the nexthop is the local machine's current nexthop. All routing information advertised by and received from a test peer is discarded, and all BGP-advertiseable routes are sent back to the test peer. Metrics from BGP-derived routes are forwarded in the advertisement; otherwise, no metric is included.

Specifying group parameters

The **`group`** statement can contain one or more of the following options (listed alphabetically):

`interface` `interface_list`

> (routing and confed groups)
>
> The *`interface_list`* provides a list of interfaces whose routes can be used to resolve BGP nexthops. By default, the nexthop in BGP routes advertised to type routing peers are the original nexthop that was specified in the arriving BGP UPDATE packet. The *`interface_list`* can optionally provide a list of interfaces whose routes are carried through the IGP for which third-party nexthops can be used instead.

`ignorefirstashop`

(external group)

Routers known as Route Servers, are capable of propagating routes without appending their own AS to the AS Path. By default, GateD drops such routes. Specifying **`ignorefirstashop`** on either the group statement or peer clause disables this feature. This option should only be used if it is positively known that the peer is a route server and not a normal router.

`lcladdr, outdelay, metrout`

(internal, routing, and confed groups)

These options must be set in the **`group`** statement, not on a per-peer basis, for group types **`confed`**, **`internal`** and **`routing`**. If these options are set on the **`peer`** statement, they must equal the values set on the corresponding group statement. For group types **`internal`** and **`routing`**, **`lcladdr`** is required if you want to use the loopback alias as the router ID.

`med`

(external group)

By default, any metric (that is, Multi_Exit_Disc (MED)) received on a BGP connection is ignored. If it is desired to use MEDs in routing computations, the **`med`** option must be specified on the group. 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 option.

`nexthopself`

(external, routing, confed, and internal groups)

If **`nexthopself`** is not specified, the router advertises externally learned routes with its original (external) nexthop. The **`nexthopself`** option causes the router to advertise itself as the next hop for externally learned routers. The address used for the nexthop is the local address on the shared network. If this option is used, the result can be that inefficient routes are followed. This can be necessary where the routers on the “shared” medium do not have full connectivity to each other.

`proto` *`proto_list`*

(routing and confed groups)

The *`proto_list`* names the interior protocol to be used to resolve BGP route nexthops, and can be the name of one or more IGPs in the configuration, including **`static`**, **`ospf`**, **`direct`**, **`bgp`**, and **`rip`**. The **`all`** keyword can also be used, though it is not recommended. By default, the nexthop in BGP routes advertised to type routing peers are the original nexthop that was specified in the arriving BGP UPDATE packet.

`subgroup` *`integer`*

(external, routing, confed, and internal groups)

The **`subgroup`** option designates different subgroups of BGP peers within the same AS (designated by the group statement). It works in conjunction with the subgroup identifier on the import and/or export policy configuration using the export or import clause. Essentially, the designation of the mathematical group (the group can have at minimum a membership of one peer statement) allows for the extensibility and flexibility for "policy per peer."

`reflector-client`

(internal, routing, and confed groups)

The **`reflector-client`** option specifies that GateD acts as a route reflector for this group. All routes received from any group member are sent to all other internal neighbors, and all routes received from any other internal neighbors are sent to the reflector clients. Because the route reflector forwards routes in this way, the reflector-client group need not be fully meshed.

If the **`no-client-reflect`** option is specified, routes received from reflector clients are only sent to internal neighbors that are not in the same group as the sending reflector client. In this case, the reflector-client group should be fully meshed. In all cases, routes received from normal internal peers are sent to all reflector clients.

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

```
export proto bgp as 2 {

      proto bgp as 2 {all;}; # for reflection

      # other exports

};
```

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

`send-community` `bgp_peer_stmts`

The **`send-community`** value specifies that any community tags belonging to routes to be exported to this group's peers are to be sent. If **`send-community`** is not specified in the BGP statement, community tags are exported by default.

`send-no-community` `bgp_peer_stmts`

The **`send-no-community`** value specifies that any community tags belonging to routes to be exported to this group's peers are removed and are not sent. If **`send-no-community`** is not specified, the default is **`send-community`**.

Peer parameters

Within a group, BGP peers can be configured in one of two ways. They can be explicitly configured with a **`peer`** statement, or implicitly configured with the **`allow`** statement. Both are described as follows:

`allow`

The **`allow`** clause allows **`peer`** connections from any addresses in the specified range of network and mask pairs. 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 detail on specifying the network/mask pairs, see "Route filtering" on page 3-53.

`peer` *`host`*

A **`peer`** clause configures an individual peer. Each peer inherits all parameters specified on a group as defaults. Many defaults can 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.`** **`allow`** is used to specify a set of address masks. If GateD receives a BGP connection request from any address in the set specified, it accepts it and set up a peer relationship.

For additional information on setting peer options, see “Group and peer parameters” on page 3-31.

Specifying peer parameters

The BGP **`peer`** subclause allows the following options, which can also be specified on the **`group`** clause. All are optional.

`metricout` *`metric`*

If specified, this metric can be used on all routes sent to the specified peer(s). The metric hierarchy is as follows, starting from the most preferred:

- Metric specified by export policy
- Peer-level metricout
- Group-level metricout
- Default metric

For group types **`internal`**, **`confed`**, and **`routing`**, set this option on the **`group`** clause instead of on the **`peer`** clause.

`localas` *`autonomous_system`*

Identifies the autonomous system that GateD is representing to this group of peers. The default is that which has been set globally in the *`autonomous_system`* statement.

`nogendefault`

Prevents GateD from generating a default route when BGP receives a valid update from its neighbor. The default route is only generated when the **`gendefault`** option is enabled.

`ignorefirstashop`

Disable dropping of routes from peers that do not insert their own AS number into the AS Path. This option should only be used if it is positively known that the peer is a route server and not a normal router.

`gateway` *`gateway`*

If a network is not shared with a peer, **`gateway`** specifies a router on an attached network to be used as the nexthop router for routes received from this neighbor. The `gateway` parameter can also be used to specify a nexthop for peers that are on shared networks. This can be used to ensure that third-party nexthops are never accepted from a given peer, by specifying that peer's address as its own gateway. This parameter is not needed in most cases.

`precedence` *`precedence`*

Specifies the precedence of routes learned from this particular BGP peer. This precedence overrides the precedence set on the BGP statement. This precedence can be overridden by the import policy. This keyword can also be stated as `protocol-precedence` or `proto-precedence`.

`preference` *`preference`*

Specifies the preference used for routes learned from these peers. This can differ from the default BGP preference set in the **`bgp`** statement, so that GateD can prefer routes from one peer, or group of peers, over others. This preference can be explicitly overridden by import policy. This keyword can also be stated as **`pref`**.

`lcladdr` *`local_address`*

Specifies the 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 the **`gateway`** parameter is used. A session with an external peer can only be opened when an interface with the appropriate local address, through which the peer or gateway address is directly reachable, is operating. For other types of peers, a peer session is maintained when any interface with the specified local address is operating. In either case, incoming connections are only recognized as matching a configured peer if they are addressed to the configured local address. For group types **`internal`** and **`routing`**, set this option on the `group` clause. For group type **`routing`**, it is advisable to set the **`lcladdr`** to a non-physical interface, such as a loopback interface.

`holdtime` *`time_seconds`*

Specifies the BGP holdtime value, in seconds, to use when negotiating the connection with this peer. If GateD does not receive a keepalive, update, or notification message within the number of seconds specified in the Hold Time field of the BGP Open message, then the BGP connection is closed. The value must be at least three minutes (180 seconds).

`version` *`version_number`*

Specifies the version of the BGP protocol to use with this peer. If not specified, the highest supported version is used first and version negotiation is attempted. If it is specified, only the specified version is offered during negotiation. Currently supported versions are 2, 3 and 4.

`passive`

If this option is used, GateD never tries to open a BGP connection with this peer or group. Instead, it waits for the peer to initiate a connection. This option was introduced to handle a problem in BGP3 and earlier, where two peers can both attempt to initiate a connection at the same time. This problem is fixed in the BGP4 protocol, so **`passive`** is not needed with BGP4 sessions. If it is applied to both sides of a peering session, **`passive`** prevents the session from ever being established. For this reason, and because it is generally not needed, do not use **`passive`**.

`sendbuffer` *`buffer_size`*
`recvbuffer` *`buffer_size`*

Controls the amount of send and receive 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. These parameters are not needed on a well-functioning systems.

`outdelay` *`time_seconds`*

Used to dampen route fluctuations. **`outdelay`** is the number of seconds a route must be present in the GateD routing database before it is exported to BGP. The default value for each is 0, meaning that these features are disabled. For group types Internal and Routing, set this option on the **`group`** clause.

`keep all`

Used to retain routes learned from a peer even if the routes' AS paths contain one of our exported AS numbers.

`show-warnings`

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 ignored.

`noaggregatorid`

Causes GateD to specify the routerid in the aggregator attribute as zero (instead of its routerid) in order to prevent different routers in an AS from creating aggregate routes with different AS paths.

`keepalivesalways`

Causes GateD to always send keepalives, even when an update could have correctly substituted for one. This allows interoperability with routers that do not completely obey the protocol specifications on this point.

`v3asloopokay`

By default GateD does not advertise routes whose AS path is looped, that is, with an AS appearing more than once in the path, to version 3 external peers. Setting this flag removes this constraint. Ignored when set on internal groups or peers.

`nov4asloop`

Prevents routes with looped AS paths from being advertised to version 4 external peers. This can be useful to avoid advertising such routes to peer that would incorrectly forward the routes on to version 3 neighbors.

`ascount` *`count_number`*

`count` is the number of times the GRF inserts its own AS number when it sends the AS path to an external neighbor. Legal values are 1.25, inclusive, the default is 1.

Higher values are typically used to bias upstream neighbors' route selection. All things being equal, most routers prefer to use routes with shorter AS Paths. Using **`ascount`**, the AS Path that the GRF sends can be artificially lengthened. The **`ascount`** supersedes the **`nov4asloop`** option; regardless of whether **`nov4asloop`** is set, the GRF still sends multiple copies of its own AS if the **`ascount`** option is set to something greater than one. If the value of **`ascount`** is changed and GateD is reconfigured, routes are not sent to reflect the new setting. If this is desired, it is necessary to restart the peer session by commenting out the peer or group, reconfiguring, and then uncommenting and reconfiguring again, or by restarting GateD.

`throttle` *`count_second`*

Limits the number of UPDATE packets to *`count`* per second. Non-GRF routers, when heavily overburdened, will timeout BGP peering sessions because they cannot keep up with the processing of packets forwarded by the GRF.

`allow bad routerid`

The BGP specification specifically requires that a BGP router choose a reasonable value (one of its IP addresses) as its router ID. If a BGP OPEN message is received with an unreasonable router ID, the specification requires that an error message be sent, and the BGP connection closed (see RFC 1771, section 6.2). GateD obeys this requirement.

Apparently, some non-GateD BGP implementations sometimes decide to send 0.0.0.0 as their router ID. Lucent does not recommend this, not only because it violates the BGP specification, but because some elements of the BGP protocol assume that all router IDs are globally unique. Proper router ID assignment ensures this, but if two routers in the same AS are allowed to send router ID 0.0.0.0, routing problems results, especially if route reflection is in use. In particular, two routers using identical router IDs, whether 0.0.0.0 or otherwise, do not have each other's routes reflected to them.

Despite this, GateD provides a way to disable router ID checking, by using the **`allow bad routerid`** peer option. Lucent strongly discourages using this command; it is provided only as a means of allowing interoperation with a faulty router for a short period of time until the faulty router can be fixed or replaced.

`logupdown`

Causes a message to be logged through the `syslog` mechanism whenever a BGP peer enters or leaves the established state.

`ttl` *`ttl`*

By default, GateD sets the IP **`ttl`** for local peers to one and the **`ttl`** for non-local peers to 255. This option mainly is provided when attempting to communicate with improperly functioning routers that ignore packets sent with a **`ttl`** of one. Not all kernels allow the **`ttl`** to be specified for TCP connections.

`traceoptions` *`trace_options`*

Specifies the tracing options for this BGP neighbor. By default these are inherited from group or BGP global trace options. See "Trace statement" on page 3-6 and the BGP-specific tracing options in the next section.

`nexthopself`

Causes the nexthop in route advertisements sent to this peer or group of peers to be set to the address that the peer is using for its peering session with this GRF even if it would normally be possible to send a third-party nexthop. If this option is used, the result can be that inefficient routes are followed. But this can be necessary where the routers on the "shared" medium do not really have full connectivity to each other.

Note: You can only specify **`nexthopself`** for **`group type external`** peers.

Tracing options

The **`state`** option works with BGP, but does not provide true state transition information.

Packet tracing options that can be modified with **`detail`**, **`send`**, and **`recv`** are as follows:

`packets`

All BGP packets.

`open`

BGP **`open`** packets that are used to establish a peer relationship.

`update`

BGP **`update`** packets that are used to pass network reachability information.

`keepalie`

BGP **`keepalive`** packets that are used to verify peer reachability.

`all`

Additional useful information, including additions/changes/deletions to the GateD routing table.

Recovering interfaces

If an interface goes down and comes back up and there is policy associated with that interface, GateD must be restarted because the interface policy is not recovered.

The policy can be recovered by manually executing the **`gdc reconfig`** command, which rereads the `/etc/gated.conf` file and does not stop or restart GateD.

Weighted route dampening statement

Currently, only routes learned through BGP are subject to weighted route dampening although no protocols announce suppressed routes.

Note: The weighted route dampening configuration statement is not within the BGP statement, but is a separate and distinct configuration conceptually, much like interface or kernel statements.

Syntax

The **weighted route dampening** statement syntax is as follows:

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

Options

The weight route dampening statement can contain one or more of the following options:

dampen-flap {}

When only **dampen-flap {};** is specified in the route dampening statement, then the following default values are used:

```
suppress-above = 3.0;
reuse-below = 2.0;
max-flap = 16.0;
unreach-decay = 900;
reach-decay = 300;
keep-history = 1800;
```

suppress-above *metric*

This is the value of the instability metric at which route suppression takes place. A route is not installed in the forwarding information base (FIB), or announced even if it is reachable during the period that it is suppressed.

reuse-below *metric*

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

max-flap *metric*

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

`reach-decay` *`time_seconds`*

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

`unreach-decay` *`time_seconds`*

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

`keep-history` *`time_seconds`*

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

ICMP statement

On systems without the BSD routing socket, GateD listens to ICMP messages received by the system. Currently, GateD does processing only on ICMP redirect packets; more functionality can be added in the future, such as support for the router discovery messages. Processing of ICMP redirect messages is handled by the **`redirect`** statement.

Currently, the only reason to specify the **`icmp`** statement is to be able to trace the ICMP messages that GateD receives.

Syntax

The **`icmp`** statement syntax is as follows:

```
icmp {
    traceoptions trace_options ;
}
```

Option

The **`icmp`** statement can contain the following option:

`traceoptions` *`trace_options`*

Specifies the tracing options for ICMP. See “Trace statement” on page 3-6 and the ICMP-specific tracing options for more information.

Tracing options

Packet tracing options that can be modified with **`detail`** and **`recv`** are as follows:

`packets`

All ICMP packets received.

`redirect`

Only ICMP redirect packets received.

`routerdiscovery`

Only ICMP Router Discovery packets received.

`info`

Only ICMP informational packets, which include mask request/response, info request/response, echo request/response and time stamp request/response.

`error`

Only ICMP error packets, which include time exceeded, parameter problem, unreachable and source quench.

Router discovery protocol

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.

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

Router discovery server

The Router Discovery Server runs on routers and announces their existence to hosts. It does this 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 router's addresses on a specific 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 responds 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 is 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 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.

Router discovery client

The client portion runs on the hosts. A host listens for Router Advertisements through the all-hosts multicast address (224.0.0.1), if IP multicasting is available and enabled, or on the interface's broadcast address. When starting up, or when reconfigured, a host can send a few Router Solicitations to the all-routers multicast address, 224.0.0.2, or the interface's broadcast address.

When a Router Advertisement with non-zero lifetime is received, the host installs a default route to each of the advertised addresses. If the preference is **`ineligible`**, or the address is not on an attached interface, the route is marked unusable but retained. If the preference is usable, the metric is set as a function of the preference such that the route with the best preference is used. If more than one address with the same preference is received, the one with the lowest IP address is 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 routers learned from ICMP redirects pointing to these addresses are deleted. The same happens when a Router Advertisement is not received to refresh these routes before the lifetime expires.

Router discovery server statement

The router discovery server runs on routers and announces their existence to hosts. See the previous pages for information about the router discovery server. The following is information regarding the router discovery server statement:

Syntax

The Router Discover server statement syntax is as follows:

```
routerdiscovery server  ( yes | no | on | off ) [ {
       traceoptions trace_options ;
       interface interface_list
              [ minadvinterval time_seconds ] |
              [ maxadvinterval time_seconds ] |
              [ lifetime time ]
              ;
       address interface_list
              [ advertise ] | [ ignore ] |
              [ broadcast ] | [ multicast ] |
              [ ineligible ] | [ preference preference ]
              ;
} ] ;
```

Options

The Router Discovery server statement can contain one or more of the following options:

`traceoptions` *`trace_options`*

Specifies the Router Discovery tracing options. See “Trace statement” on page 3-6 and the Router Discovery specific tracing options later in this section for more information.

`interface` *`interface_list`*

Specifies the parameters that apply to physical interfaces. Note a slight difference in convention from the rest of GateD; **`interface`** specifies a list of physical interfaces, such as `le0, ef0`, and `en1`, while **`address`** specifies a list of IP addresses. One or more of the following parameters must be provided for the interface:

`maxadvinterval` *`time_seconds`*

The maximum time allowed between sending broadcast or multicast Router Advertisements from the interface. Must be no less than four seconds and no more than 30 minutes or 1800 seconds. The default is 10 minutes or 600 seconds.

`minadvinterval` *`time_seconds`*

The minimum time allowed between sending unsolicited broadcast or multicast Router Advertisements from the interface. Must be no less than three seconds and no greater than **`maxadvinterval`**. The default is **`0.75 * maxadvinterval`**.

`lifetime` *`time_seconds`*

The lifetime of addresses in a Router Advertisement. Must be no less than **`maxadvinterval`** and no greater than 2:30:00 (two hours, thirty minutes or 9000 seconds). The default is **`3 * maxadvinterval`**.

`address` *`interface_list`*

Specifies the parameters that apply to the specified set of addresses on this physical interface. Note a slight difference in convention from the rest of GateD; **`interface`** specifies a list of physical interfaces (such as `le0`, `ef0,` and `en1`), while **`address`** specifies a list of IP addresses.

One or more of the following parameters must be provided for the address:

`advertise`

Specifies that the specified address(es) should be included in Router Advertisements. This is the default.

`ignore`

Specifies that the specified address(es) should not be included in Router Advertisements.

`broadcast`

Specifies that the given address(es) should be included in a broadcast Router Advertisement because this system does not support IP multicasting, or some hosts on attached network do not support IP multicasting. It is possible to mix addresses on a physical interface such that some are included in a broadcast Router Advertisement and some are included in a multicast Router Advertisement. This is the default if the router does not support IP multicasting.

`multicast`

Specifies that the given address(es) should only be included in a multicast Router Advertisement. If the system does not support IP multicasting the address(es) are not be included. If the system supports IP multicasting, the default is to include the address(es) in a multicast Router Advertisement if the given interface supports IP multicasting. If not, the address(es) are included in a broadcast Router Advertisement.

`preference` *`preference`*

The degree of preference of the address(es) as a default router address, relative to other router addresses on the same subnet. This is a 32-bit, signed, two's-complement integer, with higher values meaning more preferable. Note that hex `80000000` can only be specified as **`ineligible`**. The default is 0.

`ineligible`

Specifies that the given address(es) are assigned a preference of hex 80000000. That means that it is not eligible to be the default route for any hosts.

This is useful when the address(es) should not be used as a default route, but are given as the nexthop in an ICMP redirect. This allows the hosts to verify that the given addresses are up and available.

Router discovery client statement

For more information on the router discovery client, refer to the previous page. The following is information regarding the router discovery client statement:

Syntax

The router discovery client statement syntax is as follows:

```
routerdiscovery client  ( yes | no | on | off ) [ {
       traceoptions trace_options ;
       precedence precedence ;
       interface interface_list
              [ enable ] | [ disable ]
              [ multicast ] | [ broadcast ]
              [ quiet ] | [ solicit ]
              ;
} ] ;
```

Options

The router discovery client statement can contain one or more of the following options:

`traceoptions` *`trace_options`*

Specifies the tracing options for Router Discovery. See the “Trace statement” on page 3-6 and the Router Discovery specific tracing options later in this section for more information.

`precedence`

Specifies the precedence of all Router Discovery default routes. The default is 55.

`interface` *`interface_list`*

Specifies the parameters that apply to physical interfaces. Note a slight difference in convention from the rest of GateD; **`interface`** specifies just physical interfaces (such as `le0`, `ef0`, and `en1`). The Router Discovery Client has no parameters that apply only to interface addresses. One or more of the following parameters is available for the **`interface`**:

`enable`
Specifies that Router Discovery should be performed on the specified interface(s). This is the default.

`disable`
Specifies that Router Discovery should not be performed on the specified interface(s).

`multicast`
Specifies that Router Solicitations should be multicast on the specified interface(s). If IP multicast is not available on this host and interface, no solicitation is performed. The default is to multicast Router Solicitations if the host and interface support it; otherwise Router Solicitations are broadcast.

`broadcast`
Specifies that Router Solicitations should be broadcast on the specified interface(s) because the interface or remote system does not support IP multicasting. This is the default if the local interface does not support IP multicasting.

`quiet`
Specifies that no Router Solicitations are sent on this interface, even though Router Discovery is performed.

`solicit`
Specifies that initial Router Solicitations is sent on this interface. This is the default.

Tracing options

The Router Discovery Client and Server support the **`state`** trace flag, that traces various protocol occurrences.

`state`
State transitions.

The Router Discovery Client and Server do not directly support any packet tracing options. Tracing of router discovery packets is enabled through the ICMP statement.

Kernel statement

While the kernel interface is not technically a routing protocol, it has many characteristics of one, and GateD handles it the same manner. The routes GateD chooses to install in the kernel forwarding table are those that are actually 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. This does not present a problem for older routing protocols, for example, RIP, that are not particularly time critical and do not easily handle very large numbers of routes. The newer routing protocols, for example, OSPF, 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 installing large numbers of routes, 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 following subsection, but normally the default parameters provide the proper functionality.

During normal shutdown processing, GateD normally deletes all the routes it has installed in the kernel forwarding table, except for those marked with `retain`. Optionally, GateD can leave all routes in the kernel forwarding table by not deleting any routes. In this case, changes are made to ensure that routes with a `retain` indication are installed in the table. This is useful on systems with large numbers of routes, as it prevents the need to re-install the routes when GateD restarts. This can greatly reduce the time it takes to recover from a restart.

Forwarding tables and routing tables

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

Syntax

The **`kernel`** statement syntax is as follows:

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

```
            ;
        background
             [ limit number ]
             [ priority flash | higher | lower ]
             ;
        traceoptions trace_options ;
} ;
```

Options

The **kernel** statement can contain one or more of the following options:

options *option_list*

Configure kernel options. The valid options are as follows:

nochange

On systems supporting the routing socket, this option ensures that only deletes and adds change operations are performed; no other change operations are allowed. This option 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 from the kernel forwarding table that do not have a `retain` indication. The **noflushatexit** option prevents route deletions at shutdown. Instead, routes are changed and added to make sure that all the routes marked with `retain` get installed.

This option is useful on systems with thousands of routes. Upon start-up, GateD notices which routes are in the kernel forwarding table and does not add them back.

routes *number*

On some systems, kernel memory is at a premium. By using this option, a limit can be placed on the maximum number of routes GateD installs in the kernel. Normally, GateD adds/changes/deletes routes in interface/internal/external order. That is, it queues interface routes first, followed by internal routes, followed by external routes, and processes the queue from the beginning. If this option is specified and the limit is reached, GateD does two scans of the list. On the first scan it does deletes, and also deletes all changed routes, turning the queued changes into adds. It then rescans the list doing adds in interface/internal/external order until it hits the limit again. This operation tends to favor internal routes over external routes. The default is not to limit the number of routes in the kernel forwarding table.

flash

When routes change, the process of notifying the protocols is called a flash update. The kernel forwarding table interface is the first to be notified. Normally, a maximum of 20 interface routes can be processed during one flash update. **flash** allows tuning of the following options:

limit *number_routes*

Specifies the maximum number of routes that can be processed during one flash update. The default is 20. A value of -l causes all pending route changes of the specified type to be processed during the flash update.

`type interface | interior | all`

Specifies the type of routes that are processed during a flash update. The default is **`interface`**; it specifies that only interface routes are installed during a flash update. The **`interior`** keyword specifies that interior routes are also installed. The **`all`** keyword specifies that exterior routes are also installed as.

Specifying **`flash limit -1 all`** causes all routes to be installed during the flash update; this mimics the behavior of previous versions of GateD.

`background`

Normally, only interface routes are 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. Normally, 120 routes are installed at a time to allow other tasks to be performed and this background processing is done at lower priority than flash updates. background allows tuning of the following options:

`limit` *`number_routes`*

Specifies the number of route that can be processed at during one batch. The default is 120.

`priority flash | higher | lower`

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`**. Specifying **`higher`** means that all pending kernel updates must be finished before processing any flash updates. Specifying **`lower`** means that all pending flash updates must be finished before processing any kernel updates.

Tracing options

While the kernel interface is not technically a routing protocol, in many cases it is handled as one. The following two options are entered from the command line because the code that uses them is executed before the trace file is parsed:

`symbols`

Symbols read from the kernel, by `nlist()` or similar interface.

`iflist`

Interface list scan. This option is useful when entered from the command line as the first interface list scan is performed before the configuration file is parsed.

The following tracing options can only be specified in the configuration file. They are not valid from the command line:

`remnants`

Routes read from the kernel when GateD starts.

`request`

Requests by GateD to Add/Delete/Change routes in the kernel forwarding table.

The following general option and packet-tracing options only apply on systems that use the routing socket to exchange routing information with the kernel. They do not apply on systems that use the old BSD4.3 ioctl() interface to the kernel.

`info`

Informational messages received from the routing socket, such as TCP loss, routing lookup failure, and route resolution requests. GateD does not currently do processing on these messages; rather it just logs the information if requested.

Packet tracing options that can be modified with **`detail`**, **`send`** and **`recv`**:

`routes`

Routes exchanged with the kernel, including Add/Delete/Change messages and Add/Delete/ Change messages received from other processes.

`redirect`

Redirect messages received from the kernel.

`interface`

Interface status messages received from the kernel. These are only supported on systems with networking code derived from BSD 4.4.

`other`

Other messages received from the kernel, including those mentioned in the **`info`** keyword explanation previously.

Static statement

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

Syntax

The **`static`** statement syntax is as follows:

```
static {
       ( host host ) | default |
        ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
            gateway gateway_list
            [ interface interface_list ]
            [ precedence precedence ]
            [ retain ]
            [ reject ]
            [ blackhole ]
            [ noinstall ] ;
        ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
            interface interface
            [ precedence precedence ]
            [ retain ]
            [ reject ]
            [ blackhole ]
            [ noinstall ] ;
} ;
```

Options

The **`static`** statement can contain one or more of the following options:

```
host host gateway gateway_list ( network [ ( mask mask )
| ( ( masklen  | / ) number ) ] )  default gateway gateway_list
```

This is the most general form of the static statement. It defines a static route through one or more gateways. Static routes are installed when one or more of the gateways listed are available on directly attached interfaces. If more than one eligible gateway is available, these are limited by the number of multipath destinations supported.

The **`static`** statement can contain one or more of the following options:

`interface` *`interface_list`*

When this parameter is specified, gateways are only considered valid when they are on one of these interfaces. See “Interface statement” on page 3-12 for the description of the *`interface_list`*.

`precedence` *`precedence`*

This option selects the precedence of this static route. The precedence controls how this route competes with routes from other protocols. The default precedence is 60. This keyword can also be stated as **`protocol-precedence`** or **`proto-precedence`**.

`retain`

Normally GateD removes all routes except interface routes from the kernel forwarding table during a graceful shutdown. The **`retain`** option can be used to prevent specific static routes from being removed. This is useful to ensure that some routing is available when GateD is not running.

`reject`

Instead of forwarding a packet like a normal route, **`reject`** routes cause packets to be dropped and `unreachable` messages to be sent to the packet originators. Specifying this option causes the route to be installed as a reject route. Not all kernel forwarding engines support reject routes.

`blackhole`

A **`blackhole`** route is the same as a **`reject`** route except that unreachable messages are not supported.

`noinstall`

Normally, the route with the lowest precedence is installed in the kernel forwarding table, and is the route exported to other protocols. When **`noinstall`** is specified on a route, it is not installed in the kernel forwarding table even if it has the lowest precedence, and is therefore active.

Even if a route is not installed, it is available for other uses by GateD. In particular, the route can be exported to other protocols, for example, OSPF ASE, and can be used as a contributing route in the formation of aggregates. Additionally, an active but not installed route can be used to determine if a BGP route’s nexthop is reachable, thus permitting the BGP route itself to be installed.

`(` *`network`* `[ (` **`mask`** *`mask`* `) | ( (` **`masklen`** `| / )` *`number`* `) ] )`

`interface` *`interface`*

This form defines a static interface route that is used for primitive support of multiple network addresses on one interface. The **`precedence`**, **`retain`**, **`reject`**, **`blackhole`**, and **`noinstall`** options are the same as described previously.

Control statements

Control statements control routes that are imported from routing peers and routes that are exported to these peers. These are the final statements to be included in the `/etc/gated.conf` file.

The control statements are as follows:

Route Filtering
Matching AS Paths
Import statement
Export statement
Aggregate statement
Generate statement

Route filtering

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

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **`define`** filter statement. See “Global filters” on page 3-55 for more information.

The action taken when no match is found depends on the context. For instance, `import` and `export` route filters assume an **`all restrict;`** at the end of a list.

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

Syntax

The route filtering syntax is as follows:

`network` `mask` *`mask`* `[` **`exact`**`|`**`refines`**`|`**`between`** *`number`* **`and`** *`number`* `]`

`network` **`masklen`** `|/` *`number`* `[` **`exact`** `|` **`refines`** `|` **`between`** *`number`* **`and`** *`number`* `]` **`all default host`** *`host`*

The syntaxes are all the possible formats for a route filter. Not all of these formats are available in all places. For example, the **`host`** and **`default`** formats are not valid for the **`martians`** statement.

In most cases, you can specify additional options relevant to the context of the filter. For example, on a **`martians`** statement it is possible to specify the **`allow`** option. On an **`import`** statement you can specify a preference. On an **`export`** statement you can specify a metric.

Options

The filtering statement can contain one or more of the following options:

network **mask** *mask* [**exact** | **refines** | **between** *number* **and** *number*]
network (**masklen** | **/**) *number* [**exact** | **refines** | **between** *number* **and** *number*]

Matching requires both an address and a mask, except for **all**, **default**, and **host**. These forms vary in how the mask is specified. In the first, the mask is explicitly specified. In the second, the mask is specified by the number of contiguous one bits.

If no additional options are specified, any destination that falls in the range given by the network and mask is matched. The mask of the destination is ignored.

Modifiers

The three following optional modifiers cause the mask of the destination to be considered:

exact

This parameter specifies that the mask of the destination must match the supplied mask exactly. This is used to match a network, but no subnets or hosts of that network.

refines

Specifies that the mask of the destination must be more specic, that is, longer than the filter mask. This is used to match subnets and/or hosts of a network, but not the network.

between *number* **and** *number*

Specifies that the mask of the destination must be as or more specific (as long as or longer) than the lower limit (the first *number* parameter) and no more specific (as long as or shorter) than the upper limit (the second *number* parameter). The **exact** and **refines** options are both special cases of **between**.

More than one filter can be specified with a given network and mask, as long as the filters differ in their modifiers. In the case of two otherwise identical filters with the between modifier, the ranges given by the filters must not overlap.

If two filters differ only in terms of their modifiers, the filters are matched in the following order: **exact, between, refines**, no modifier. For example, a filter with **exact** specified is matched before an eligible filter with **between**.

all

This entry matches anything. It is equivalent to the following:

```
0.0.0.0 mask 0.0.0.0
```

default

Matches the **default** route. To match, the address must be the default address and the mask must be all zeros. This is equivalent to the following:

```
0.0.0.0 mask 0.0.0.0 exact
```

host *host*

Matches a specific host route. To match, the address must exactly match the specified *host* and the network mask must be a host mask, that is, all ones. This is equivalent to the following:

```
host mask 255.255.255 exact
```

Global filters

Traditionally, when configuring a route filter, a list of aspath statements or list of aggregates is used as the import or export policy. Either a filter must be created for each import or export statement, or, if the filter or list is to be used multiple times or “globally”, a **`%include`** statement is used for each import or export statement.

Using this method meant that managing a policy configuration required one of the following:

- Edit numerous configuration files by hand.
- Because of the nature of the **`%include`** statement, the same filters or lists are parsed, and redundant structures for filters and lists are created in memory.

This “managing policy by hand” is unacceptable. The CPU and memory costs associated with **`%include`** are so large that restarting or reconfiguring GateD was causing delays of processing route update information beyond reasonable expectations.

To alleviate these issues the **`define`** statement is used to reference filters or lists. When used in conjunction with **`%include`** statements, the ease of editing a single file, and the use of CPU and memory is optimized. Also, depending on the complexity of import or export statements, the syntax is compressed to a very simple statement.

Defining global filters and list

The following paragraphs explain the different ways of defining a global list or filters.

Each **`define`** statement has an option after the *`filter_name`*, for example, { *`import_list_inet`* }. To use these options, see the following sections for more information:

- For { *`import_list_inet`* }, any of the options in “Route filtering” can be used between the { }.
- For { *`prop_source_list_inet`* }, any of the options in the protocol-specific sections found under “Specifying the source” on page 3-72 can be used.
- For { *`aggregate_list_inet`* }, any of the options in the aggregation syntax can be used; see page 3-76.

The following is the **`define`** statement that names an import filter. An import filter is a list of prefixes as described in the `import` (see page 3-64) and `filter` (see page 3-53) statements. The syntax of this type of `define` statement is as follows:

```
define import filter “filter_name” { import_list_inet }
```

The following is the **`define`** statement that names an export filter for the specified protocol. An export filter is a list of prefixes as described in the **`export`** and **`filter`** statement. Export filters are used only when exporting to the specified protocol. The syntax of this type of `define` statement is as follows:

```
define proto export filter “filter_name” {import_list_inet }
```

The following is the `define` statement that names a generic filter. Generic filters can only permit or restrict prefixes. See “Route filtering” on page 3-53 for more information on configuring filters. Preferences, metrics, and so on cannot be set. Generic filters can be used for importing or exporting to any protocol. The syntax of this type of **`define`** statement is as follows:

```
define filter "filter_name" { import_list_inet }
```

The following is the `define` statement that names a group of export sources:

```
define export list "filter_name" { prop_source_list_inet }
```

The following is the **`define`** statement that names groups of aggregation sources. It is similar to the, but is used for aggregation.The syntax of this type of **`define`** statement is as follows:

```
define aggregate list "filter_name" { aggregate_list_inet }
```

Note: Filter names may include spaces, punctuation and other characters other than double quotes (“). Also, you cannot use quotes or backslashes.

The following examples show how to use the **`define`** statement:

Example 1:

```
define filter "filter-2" {
default restrict;
};

define export list "exp-list-80" {
proto rip restrict;
proto ospfase restrict;
proto bgp aspath .* 6999 origin any "filter-2";
};

export proto bgp as foo "exp-list-80";
```

Example 2:

```
define bgp export filter "bgp-filter" {
all metric 10;
};

export proto bgp as bar "bgp-filter";
```

Matching AS paths

An AS path is a list of autonomous systems that routing information has passed through to get to this router, and an indicator of the origin of this information. This information can be used to prefer one path to a destination network over another. The primary method for doing this with GateD is to specify a list of patterns to be applied to AS paths when importing and exporting routes. Each autonomous system through which a route passes prepends its AS number to the beginning of the AS path.

The origin information details the completeness of AS path information. An origin of **`igp`** indicates the route was learned from an interior routing protocol and is most likely complete. An origin of **`egp`** indicates the route was learned from an exterior routing protocol that does not support AS paths and the path is most likely not complete. When the path information is definitely not complete, an origin of **`incomplete`** is used.

There is also a way to define filters and lists such that they can be referenced by a common name through a **`define`** filter statement. See “Global filters” on page 3-55 for more information.

Syntax

An AS path is matched using the following syntax:

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

This syntax specifies that an AS matching the *`aspath_regexp`* with the specified origin is matched.

AS path regular expressions

Technically, an AS path regular expression is a regular expression with the alphabet being the set of AS numbers. An AS path regular expression is composed of one or more AS path expressions. An AS path expression is composed of AS path terms and AS path operator, which are explained in the following sections.

AS path terms

An AS path term is one of the following three objects:

```
autonomous_system
.
( aspath_regexp )
```

`autonomous_system`

Is any valid autonomous system number, from one through 65534 inclusive.

.

Matches any autonomous system number.

`(` *`aspath_regexp`* `)`

Parentheses group subexpressions -- an operator such as `*` or `?` works on a single element, or on a regular expression enclosed in parentheses.

AS path operators

Note: Under release 1.4.6, the **`aspath`** operator had to be included within double quotes. While the double quotes are no longer needed, however, they are backwards compatible.

An AS path operator is one of the following:

```
aspath_term {m,n}
aspath_term {m}
aspath_term {m,}
aspath_term *
aspath_term +
aspath_term ?
(aspath_term) | (aspath_term)
null
[n1  n2 n3 ...]
```

`aspath_term` `{ m, n }`

A regular expression followed by { **`m, n`** } where **`m`** and **`n`** are both non-negative integers and m <= n, means at least **`m`** and at most **`n`** repetitions.

`aspath_term` `{ m }`

A regular expression followed by {**`m`**} (where **`m`** is a positive integer) means exactly **`m`** repetitions.

`aspath_term` `{ m, }`

A regular expression followed by {**`m,`**} (where **`m`** is a positive integer) means **`m`** or more repetitions.

`aspath_term` **`*`**

An AS path term followed by **`*`** means zero or more repetitions. This is shorthand for `{0,}`.

`aspath_term` **`+`**

A regular expression followed by **`+`** means one or more repetitions. This is shorthand for `{1,}`.

`aspath_term` **`?`**

A regular expression followed by **`?`** means zero or one repetition. This is shorthand for `{0,1}`.

`null`

Matches all things with a null aspath, as shown in the following example:

```
import proto bgp aspath (null) origin any { all; };
```

[*`n1  n2 n3 ..`*.]

Defines a set of numbers defined by n (where **`n`** is an AS number) in the brackets, and the input must match one of the numbers in the set. Ranges of numbers cannot be specified.

`(aspath_term)` `|` *`(aspath_term)`*

Matches the AS term on the left, or the AS term on the right. This is identical to a logical OR operation.

Examples

Parentheses can be used to improve readability or to group AS path terms into a subexpression. The subexpression can then be used as an aspath term with any of the operators, as shown in the following example:

```
aspath 1 3 4 *
```

The previous example translates to 1 3 (4*), with the ***** operator applied only to the **4**. If you were to place the parentheses around all the numbers, that is, (1 3 4)*, the ***** would apply to the whole subexpression, that is, 1 3 4.

The following examples show how the AS path operators are used with the AS path terms to select an AS path.

Assume the following AS path regular expression is defined:

```
aspath 7 {2,5}
```

The {**2,5**} defines the number of times that the aspath number **7** can be repeated. This translates into the following AS path expressions being selected: 7 7, 7 7 7, 7 7 7 7, and 7 7 7 7 7. In other words, the AS number 7 can be repeated at least twice and as many times as five to be selected.

Assume the following AS path regular expression is defined:

```
aspath 7 {4}
```

This translates into only the AS path 7 7 7 7 being selected.

Assume the following AS path regular expression is defined:

```
aspath 4 {3,}
```

This translates into three or more repetitions of 4 4 4 being selected.

The following AS numbers are used in the next set of examples:

1 2 3 4
5 7 1 5
4 1 3 7
1 3
4 1 3
4 1 3 7 8
3 4 5 1 3

Assume that the following AS path regular expression is defined:

```
aspath .* 1 3 .*
```

This is translated as follows: the **.*,** that can be no AS number or, one or more AS numbers followed by **1**; followed by a **3**; followed by **.*** , that can be no AS number or, one or more AS numbers. This definition would result in path expressions C, D, E, F, and G being selected.

Assume the following AS path regular expression is defined:

```
aspath .* 1 3 .+
```

This is translated as follows: the `.*,` that can be no AS number or, one or more AS numbers, that is, zero or more repetitions; followed by **1**; followed by a **3**; followed by **.+** , that can be any AS numbers, that is, one or more repetitions. This definition would result in path expressions C and F being selected.

Assume the following AS path regular expression is defined:

```
aspath .? 1 3 .?
```

This is translated as follows: the **.?**, which is optional, but can be any AS number; followed by a **1**; followed by a **3**; followed by **.?**, which is optional, but can be any AS number. This definition would result in path expressions C, D, and E being selected. Another way of viewing the use of the **?** is that it makes the AS numbers at the beginning and end optional; combining the **?** with the **.** means that the position is optional, but if the position exists, it can be any AS number.

Assume the following AS path regular expression is defined:

```
aspath (1 2 3 4) | (1 3)
```

This is translated as follows: the string of numbers "**1 2 3 4**" or "**1 3**" can be selected. This definition would result in path expressions A and D being selected. Another way of viewing the use of the **|** is that it is simply a logical OR operation.

The **|** operator is the lowest precedence operator; subexpressions are grouped before the **|** operator is applied and subexpressions are grouped from left to right. For example,” 7 | 1 6” is equivalent to “(7) | (1 6)”. That is, the “1 6” is grouped before the operator is applied. The use of parentheses is recommended to avoid ambiguity; such as someone misreading the previous example as “(7 | 1) 6”.

Assume the following AS path regular expression is defined:

```
aspath null
```

This results in the selection of items with no AS number. The **null** operator is commonly used to match routes that originate in the same AS and thus have no aspath. An example aspath expression is as follows:

```
import proto bgp aspath (null) origin any {default restrict;};
```

This statement translates to mean that if any of my IBGP peers sends a default route through IBGP, then do not install it.

The use of the **null** operator can be combined with other operators, as shown in the following example:

```
aspath 4 1 3 (7 | null)
```

This results in path expressions C and E being selected. The following is equivalent to the previous example:

```
4 1 3 7?

4 1 3 (7 | null)

( (4 1 3 7) | (4 1 3) )

4 1 3 [7 null]
```

The following AS paths are used in the next example:

3 27 54
6 27 54
12 27 54

Assume the following AS path regular expression is defined:

```
aspath [ 3 4 5 6 ] 27 54
```

The use of the numbers in the brackets means that the first position can be any of the numbers defined in the brackets. This results in path expressions A and B being selected. You cannot define a range of numbers to be used in the brackets. For example, defining the following AS path regular expression would not work:

```
aspath [3-6] 27 54
```

AS path attributes for communities

BGP updates carry a number of path attributes. Some of these, like the AS path, are required. Others are optional and may or may not appear in any given BGP update.

The **`mod-aspath`** option is used on the **`export`** statement to set a community value. The **`community`** and **`delete community`** attributes are supported. The use of **`community`** implicitly adds a community tag to a route. The use of **`delete community`** strips the associated community tag from an exported route.

The **`aspath-opt`** attributes can be used on both **`import`** and **`export`** statements to allow optional attributes to be considered when determining GateD's preference for the routes in a particular BGP update. Note, however, **`aspath-opt`** has different actions on each type of statement. Pattern matching via the use of **`any`** is applicable only to the **`export`** statement. The **`any`** attribute acts as a wild card.

Communities can be specified as follows:

- A community ID
- An AS community ID
- One of the distinguished special communities

When originating BGP communities, the set of communities that is actually sent is the union of the communities received with the route, those specified in group policy and those specified in export policy.

When receiving BGP communities, the update is matched only if **`all`** communities specified in **`aspath-opt`** are present in the BGP update. If additional communities are also present in the update, it is still matched.

There is a limit of 25 communities in any single policy clause.

Syntax for import

The **aspath-opt** syntax for import use is as follows:

```
aspath-opt {
    [community autonomous_system : community-id | community-id ]
    [ community no-export| no-advertise | no-export-subconfed| none ] }
```

Syntax for export

The **aspath-opt** statement also applies to routes being exported to BGP. Pattern matching using the **any** keyword before or after the colon can be used to match any ordinary community. Note that pattern matching cannot be used to match the well-known community values, even with "**any** : **any**". Those must be matched explicitly.

The **aspath-opt** syntax for export use is as follows:

```
aspath-opt {
    [community autonomous_system : community-id | community-id ]
    [ community no-export| no-advertise | no-export-subconfed| none ]
    [community any : community-id ]
    [community autonomous_system : any ]
    [community any : any ]
}
```

The **mod-aspath** statement applies to routes being exported to BGP and has a **delete** option to strip community tags from routes. The use of **community** (without **delete**) implicity adds community tags to routes. The **any** keyword may be used with **delete**. Pattern matching using the **any** keyword before or after the colon can be used to match any ordinary community. Note that pattern matching cannot be used to match the well-known community values, even with "**any** : **any**". Those must be matched explicitly.

The **mod-aspath** syntax for export use is as follows:

```
mod-aspath {
    [community (( autonomous_system : community-id )
     |community-id | no-export | no-advertise | no-export-subconfed )]
    [delete community (( autonomous_system : community-id )
     |community-id | no-export | no-advertise | no-export-subconfed )]
    [delete community autonomous_system : any ]
    [delete community any : community-id ]
    [delete community any : any ]
}
```

Options

The **`aspath-opt`** and **`mod-aspath`** statements can contain one or more of the following options:

`any`

The **`any`** keyword acts as a wild card for *`autonomous_system`* and/or *`community-id`* in community tags on exported routes.

Note: It is illegal to use the **`any`** keyword with the deprecated **`comm-split`** keyword.

`community` *`autonomous_system : community_id`*

This option associates an AS with a community. The *`autonomous_system`* part of the community should be set to the local AS unless there is a specific need to do otherwise.

`community` *`community_id`*

When a *`community_id`* is specified, the *`autonomous_system`* of the advertising router is implicitly prepended.

Note: For backward compatibility, the following statement is allowed with limited use:

```
community autonomous_system:community-id:
                    comm-split autonomous_system community-id
```

`community no-export`

This is a special community that indicates that the routes associated with this attribute must not be advertised outside a BGP AS boundary.

`community no-advertise`

This is a special community that indicates that the routes associated with this attribute must not be advertised to other BGP peers.

`community no-export-subconfed`

This is a special community that indicates that the routes associated with this attribute must not be advertised to BGP peers outside the routing domain, within a confederation.

`community none`

This is not actually a community, but rather a keyword that specifies that a received BGP update is only to be matched if no communities are present. It has no effect when originating communities; therefore it can only be used with **`aspath-opt`**.

`delete community` *`autonomous_system : community_id`*

This option associates an AS with the community tag that is to be deleted from the route.
`delete community` *`community_id`*

This option specifies that routes containing *`community_id`* have those tags deleted before forwarding. When only *`community_id`* is specified, the *`autonomous_system`* of the advertising router is implied.

`delete community no-export`

This option specifies that routes containing **`no-export`** have those tags deleted before forwarding to peers within the AS boundary.

`delete community no-export-subconfed`

This option specifies that routes containing **`no-export-subconfed`** have those tags deleted before forwarding to peers in your routing domain, within a confederation.

Import statement

The **`import`** statement controls which routes received from other systems are used by GateD. That is, it controls the importation of routes from routing protocols and installation of the routes in GateD's routing database.

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **`define`** filter statement. See “Global filters” on page 3-55 for more information.

Syntax

The following sections describe the syntax of the **`import`** statement. The format of the **`import`** statement varies depending on the source protocol, so to ensure ease of use, the syntax information is grouped by protocol. The exception is the following two sections “Installation of routes” and “Route filters”, which describe the common parts of the syntax. These two can be used with any of the rest of the protocol-specific parts of the **`import`** statement.

Controlling installation of routes

When importing, the following four options can be specified to control how matching routes are compared to determine the route that becomes the active route:

`restrict`

`precedence` *`precedence`*

`preference` *`preference`*

`localpref` *`preference`*

`restrict`

Specifies that the routes are not desired in the routing table. In some cases, this means that the routes are not installed in the routing table. In other cases, it means that they are installed with a negative preference, that prevents them from becoming active, and they are not installed in the forwarding table or exported to other protocols.

`precedence` *`precedence`*

This option is used by BGP; it is not propagated with the route. Specifies the precedence value used when comparing this route to routes from other protocols. The route with the lowest precedence is preferred. The precedence value set in the **`import`** statement overrides any precedence set for individual protocols. for example, RIP, OSPF, BGP, and so on. Lucent recommends that changes to preference be used before changes to precedence. In other words, if you can resolve the issues using **`preference`** within a routing protocol, do so. This method is preferred over changing the order of protocols through **`precedence`**.

`preference` *`preference`*

This option is used by BGP only; it is not propagated with the route. Specifies the preference value used when comparing this route to other routes from the same protocol. The route with the lowest value becomes the active route, is installed in the forwarding table, and is eligible to be exported to other protocols.

`localpref` *`preference`*

This option is used by BGP only; it is propagated with the route. Specifies the local preference value used when comparing BGP routes within the local AS. The route with the highest local preference is most preferred. The default local preference for BGP routes is 100.

Route filters

Route filters can be used at several levels within the **import** statement to further control the acceptance of route advertisements based on the destination address. See “Global filters” on page 3-55 for more information.

Importing routes from BGP syntax

The BGP portion of the **import** statement syntax is as follows:

```
import proto bgp as autonomous_system
    [ subgroup integer ] [ aspath-opt] restrict ;
import proto bgp as autonomous_system
    [ subgroup integer ] [ aspath-opt ]
    [ precedence precedence ][ preference preference ]
     [ localpref preference ] {
    route_filter  [ restrict | ( precedence precedence )
    | ( preference preference ) | ( localpref preference ) ] ;
} ;


import proto bgp aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ aspath-opt ] restrict ;
import proto bgp aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ aspath-opt ] [ precedence precedence ] [ preference
      preference ] [ localpref preference ]
     route_filter [ restrict | ( precedence precedence )
    | ( preference preference ) | ( localpref preference  } ;
```

BGP supports controlling propagation by the use of AS path regular expressions, that are documented in the section on “Matching AS paths” on page 3-57. The BGP versions 2 and 3 only support the propagation of natural networks, so the **host** and **default** route filters are meaningless. BGP version 4 supports the propagation of any destination along with a contiguous network mask.

Importing routes from BGP options

`aspath-opt`

Allows the specification of import policy based on the path attributes found in the BGP update. If multiple communities are specified in the **`aspath-opt`** option, only updates carrying all of the specified communities are matched. If **`none`** is specified, only updates lacking the community attribute are matched. See“AS path attributes for communities” on page 3-61 for more information on **`aspath-opt`**.

It is quite possible for several BGP import clauses to match a given update. If more than one clause matches, the first matching clause is used. All later matching clauses are ignored. For this reason, it is generally desirable to order import clauses from most to least specific. An import clause without an **`aspath-opt`** option matches any update with any or no communities.

`restrict`

BGP stores any routes that were rejected by not being mentioned in a route filter, or with the **`restrict`** keyword in the routing table, with a negative preference. A negative preference prevents a route from becoming active. This alleviates the need to break and reestablish a session upon reconfiguration if import policy is changed.

`localpref`

Has meaning for BGP routes that are local to the AS. Unlike the **`precedence`** and **`preference`** options, that only have meaning internally to the router, **`localpref`** is propagated along with a BGP route to other routers. Specifying **`localpref`** on the **`import`** statement changes the **`localpref`** of a BGP route that is received. The default **`localpref`** for a BGP route is 100. The **`localpref`** can also be changed when exporting BGP routes.

Importing routes from RIP/redirect syntax

The RIP/redirect portion of the **`import`** statement syntax is as follows:

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

The importing of RIP and redirect routes can be controlled by protocol, source interface and source gateway. If more than one is specified, they are processed from most general protocol to most specific gateway.

RIP does not support the use of preference to choose between routes of the same protocol. That is left to the protocol metrics. Routes that are rejected by the import policy are discarded.

Importing routes from OSPF syntax

The OSPF portion of the **`import`** statement syntax is as follows:

```
import proto ospfase [ tag ospf_tag ] restrict ;
import proto ospfase [ tag ospf_tag ]
    [ precedence precedence ] {
    route_filter  [ restrict | ( precedence  precedence ) ;
} ;
```

Due to the nature of OSPF, only the import of ASE routes can be controlled. OSPF intra-area and inter-area routes are always imported into the GateD routing table with a precedence of 10. If a tag is specified, the import clause only applies to routes with the specified tag.

It is only possible to restrict the importing of OSPF ASE routes when 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.

Like the other interior 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.

Export statement

The **`export`** statement controls which routes are advertised by GateD to other systems. The syntax of the **`export`** statement is similar to the syntax of the **`import`** statement, and the definitions of many of the parameters are identical. The main difference between the two statements is that while route importation is controlled just by source information, route exportation is controlled by both destination and source information.

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

There is also a way to define filters and lists such that they can be referenced by a "common name" through a **`define`** filter statement. See “Global filters” on page 3-55 for more information.

Syntax

The following sections describe the syntax of the **`export`** statement. The format of the **`export`** statement varies depending on the source protocol, so the syntax information is grouped by protocol. The exception to this is the following two sections. “Controlling exportation of routes” and “Route filters”, which describe the common parts of the syntax.

Controlling exportation of routes syntax

The following options can be specified to control how routes are exported. The most specific statement is the one applied to the route being exported, that is, through the **`metric`** keyword. The values that can be specified for **`metric`** depend on the destination protocol that is referenced in the following **`export`** statement. The syntax is as follows:

`restrict`

[**`all`**] **`metric`** [**`igp`**] *`metric`*

`localpref` *`preference`*

Controlling exportation of routes options

`restrict`

Specifies that nothing should be exported. If specified on the destination portion of the **`export`** statement, it specifies that nothing at all should be exported to this destination. If specified on the source portion, it specifies that nothing from this source should be exported to this destination. If specified as part of a route filter, it specifies that the routes matching that filter should not be exported.

`metric` *`metric`*

Specifies the metric to be used when exporting to the specified destination. **`all metric igp`** is used for exporting the IGP metric as a BGP MED.

Example

The following is an example that exports the OSPF metric as the BGP MED:

```
export proto bgp as foo {
                proto ospf {
                         all metric igp ;
                };
        };
```

Note: If this option is used with BGP, a BGP update is issued immediately upon the IGP metric changing.

When specifying **`all metric igp`**, instabilities within the IGP can cause substantial BGP route flap. This can be minimized by using the **`outdelay`** variable in the BGP statement.

`localpref` *`preference`*

This is used for setting the local preference to be exported to other IBGP peers. It can be set anywhere metric can be set in the **`export`** statement. In the following example, assume the AS is *`bar`* (using IBGP); it exports the OSPF routes into BGP with the **`localpref`** of 150:

```
export proto bgp as bar {
               proto ospf  localpref 150 {
                       all ;
               };
        };
```

The following example exports the RIP routes into BGP with the **localpref** of 170:

```
export proto bgp as bar {
               proto ospf{
                       all localpref 170;
               };
        };
```

Route filters

Route filters can be used at several levels within the **`export`** statement to further control the acceptance of route advertisements based on the destination address. See “Global filters” on page 3-55 for more information.

Specifying the destination

The syntax of the **`export`** statement varies depending on which protocol it is being applied. In all cases the specification of a metric is required. All protocols define a default metric to be used for routes being exported. In most cases, this can be overridden at several levels of the `export` statement. The specification of the source of the routing information being exported is described in the following sections.

Exporting to BGP syntax

The BGP portion of the **export** statement syntax is as follows:

```
export proto bgp as autonomous system
      restrict ;
export proto bgp as autonomous system [ mod-aspath ]
     [ subgroup integer ] [ metric metric ]|
      [ localpref preference] {
      export_list ;
} ;
```

Exporting to BGP is controlled by autonomous system; the same policy is applied to all routers in the AS or subgroup.

BGP metrics are 16-bit unsigned quantities. They range from 0 to 65535, inclusive, with 0 being the most attractive. While BGP version 4 actually supports 32-bit unsigned quantities, GateD does not yet support this. In BGP version 4, the metric is otherwise known as the multi-exit discriminator (MED). The MED has meaning only for external BGP routes, for example, routes advertised to other ASs.

Exporting to BGP options

localpref

For internal BGP routes. Specifying **localpref** on the export statement changes the already existing **localpref** value for a BGP route when it is sent to other networks. The default **localpref** for a BGP route is 100. The **localpref** can be changed when importing BGP routes as well. In fact, it is preferable to change **localpref** when importing rather than exporting BGP routes.

mod-aspath

Can be used to send the BGP community attribute. Any communities specified with the **mod-aspath** option are sent in addition to any received with the route or specified in the group statement. See “AS path attributes for communities” on page 3-61“ for more information on **mod-aspath**.

If no export policy is specified, only routes to attached interfaces are exported. If any policy is specified, the defaults are overridden; it is necessary to explicitly specify everything that should be exported.

Note that BGP versions 2 and 3 only support the propagation of natural networks, so the `host` and `default` route filters are meaningless. BGP version 4 supports the propagation of any destination along with a contiguous network mask.

Exporting to RIP syntax

The RIP portion of the **export** statement syntax is as follows:

```
export proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     restrict ;
export proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     [ metric metric ] {
export_list ;
} ;
```

Exporting to RIP is controlled by protocol, interface, or gateway. If an option is specified in more than one place, they are processed from most general protocol to most specific gateway. It is not possible to set metrics for exporting RIP routes into RIP. Attempts to do this are ignored.

If no export policy is specified, RIP and interface routes are exported into RIP. If any policy is specified, the defaults are overridden. It is necessary to explicitly specify everything that should be exported.

RIP version 1 (RIPv1) assumes that all subnets of the shared network have the same subnet mask so they are only able to propagate subnets of that network. RIP version 2 (RIPv2) removes that restriction and is capable of propagating all routes when not sending RIPv1-compatible updates.

Note: For RIP, to announce routes from other protocols that do not have a **metric** associated with them, a metric must be specified in the **export** statement. Just setting a default metric for RIP is not sufficient.

Exporting to OSPF syntax

The OSPF portion of the **export** statement syntax is as follows:

```
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
     restrict ;
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
     [ metric metric ] {
     export_list ;
} ;
```

Note: Keep the order of the OSPF export statements exactly as shown; reversing them causes a parser error.

It is not possible to create OSPF intra- or inter-area routes by exporting routes from the GateD routing table into OSPF. It is only possible to export from the GateD routing table into OSPF ASE routes. It is also not possible to control the propagation of OSPF routes within the OSPF protocol.

Exporting to OSPF options

`type 1 type 2`

There are two types of OSPF ASE routes: **`type 1`** and **`type 2`**. See the OSPF section in Chapter 1 for a detailed explanation of the two types. The default type is specified by the **`defaults`** subclause of the **`ospf`** statement. This can be overridden by a specification on the `export` statement.

`tag`

OSPF ASE routes also have the provision to carry a **`tag`**. This is an arbitrary 32-bit number that can be used on OSPF routers to filter routing information. See the "OSPF statement" on page 3-22 for detailed information on OSPF tags. The default tag specified by the **`ospf defaults`** statement can be overridden by a tag specified on the **`export`** statement.

Specifying the source

The export list specifies export based on the origin of a route; the syntax varies depending on the source. The following sections describe the syntaxes.

Exporting BGP routes syntax

The BGP portion of the `export` statement syntax is as follows:

```
proto bgp as autonomous_system   [ subgroup integer ]
     restrict ;
proto bgp as autonomous_system  [ subgroup integer ]
     [ metric metric  | localpref preference ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

BGP routes can be specified by source autonomous system. All routes can be exported by **`aspath`**. See "Exporting by AS path" on page 3-74" for more information.

The **`localpref`** parameter only has meaning for internal BGP routes. Specifying **`localpref`** on the **`export`** statement changes the already existing **`localpref`** value for a BGP route when it is sent to other routes. The default **`localpref`** for a BGP route is 100. The **`localpref`** can be BGP routes as well. In fact, it is preferable to change **`localpref`** when importing rather than exporting BGP routes.

Exporting RIP routes

RIP routes can be exported by protocol, source interface and/or source gateway. The RIP portion of the **`export`** statement syntax is as follows:

```
proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     restrict ;
proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     [ metric metric ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Exporting OSPF routes

Both OSPF and OSPF ASE routes can be exported into other protocols. See the following sections for information on exporting by **`tag`**. The OSPF portion of the `export` statement syntax is as follows:

```
proto ospf | ospfase restrict ;
proto ospf | ospfase [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Exporting routes from non-routing protocols

The following protocols can be exported by protocol, or by the interface of the nexthop. The protocols are as follows:

`direct`

Routes to directly-attached interfaces.

`static`

Static routes specified in a `static` statement.

`kernel`

On systems with the routing socket, routes learned from the routing socket are installed in the GateD routing table with a protocol of **`kernel`**. These routes can be exported by referencing this protocol. This is useful when it is desirable to have a script install routes with the **`route`** command and propagate them to other routing protocols.

Non-routing by interface

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

Non-routing by protocol

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

default

Refers to routes created by the **gendefault** option. Lucent reccommends that route generation be used instead.

aggregate

Refers to routes synthesized from other routes when the **aggregate** and **generate** statements are used. See “Route aggregation and generation statements” on page 3-75 for more information.

Exporting by AS path

```
proto bgp | all aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     restrict ;
proto bgp | all aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ metric metric  | localpref preference ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

When BGP is configured, all routes are assigned an AS path when they are added to the routing table. For all interior routes, this AS path specifies IGP as the origin and no ASs in the AS path. The current AS is added when the route is exported. For BGP routes, the AS path is stored as learned from BGP.

See “Matching AS paths” on page 3-57 for more information on AS path regular expressions.

Exporting by route tag

```
proto rip | ospf | all tag tag restrict ;
proto rip | ospf | alltag tag
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

Both OSPF and RIPv 2 currently support **tags**, all other protocols always have a tag of zero. The source of exported routes can be selected based on this tag. This is useful when routes are classified by tag when they are exported into a given routing protocol.

Route aggregation and generation statements

Route aggregation is a method of generating a more general route based on one or more specific routes. It is used, for example, at an autonomous system border to generate a route to a network to be advertised through BGP given the presence of one or more subnets of that network learned through an IGP.

Older versions of GateD automatically performed this function, generating an aggregate route to a natural network given an interface to a subnet of that natural network. However, that was not always the correct thing to do, and with the advent of classless inter-domain routing it is even more frequently the wrong thing to do, so aggregation must be explicitly configured. No aggregation is performed unless explicitly requested in an **`aggregate`** statement.

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

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

Route aggregation can be specified through two statements: **`aggregate`** and **`generate`**. A route built with the **`aggregate`** statement and advertised to a peer through BGP is sent with the aggregator ID and with aggregation attributes. A route built with the `generate` statement and advertised to a peer through BGP is sent without the aggregator ID and without aggregation attributes.

By default, GateD advertises all the contributing routes, in addition to the aggregate. It is necessary to explicitly filter out the contributing routes in the **`export`** statement.

The set of routes that form aggregate routes are known as contributing routes. The contributing routes are defined by the **`proto`** and/or **`route_filter`** definitions within the **`aggregate`**/**`generate`** statement. The contributing routes are ordered according to the precedence and preference that applies to them. If there is more than one contributing route of the same protocol, the route with the lower preference is ordered first. If there are contributing routes from different protocols, those with a lower precedence are ordered first. The resulting aggregate route is assigned the precedence and preference values from the contributing route which is ordered first. Lucent recommends that preference be used to control the order of the contributing routes that are learned from the same protocol before using precedence to control the order of contributing routes that are learned from different protocols.

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

The use of the **`default`** parameter is shown in the following example. Assume that the topology of this simple network contains four GRFs and one TNT. GRF3 is advertising route 10.8.17/30 to GRF1, while GRF4 is advertising route 10.8.16/30 to GRF2. GRF1 and GRF2 are then connected to the TNT box through RIP. Assume that the following code is part of the `/etc/gated.conf` file for GRF1:

```
aggregate default {
   proto direct {10.8.17/30;} ;
} ;

export proto rip metric 3 {
  proto aggregate {all} ;
} ;
```

Assume that the following code is part of the `/etc/gated.conf` file for GRF2:

```
aggregate default {
   proto direct {10.8.16/30;} ;
} ;

export proto rip metric 2 {
  proto aggregate {all} ;
} ;
```

Note: The **`aggregate`** statement must come after the protocol statements in the `/etc/gated.conf` file.

The use of the **`default`** parameter in the previous example means that if the TNT receives any packet with an unknown destination address, the packet is sent to the appropriate GRF through the **`default`** route. The default route is built, but is not put in the forwarding table. It is built for export purposes only.

Instead of using the default route, you can aggregate to a specific route (for example, 10/8) by replacing **`default`** with 10/8 that is, by using the **`masklen`** *`number / number`* parameter.

Aggregation and generation syntax

The **`aggrerate`** and **`generate`** syntax is as follows:

```
aggregate
     default  |  ( network [ ( mask mask ) | ( ( masklen
      number | / ) number ) ] )
      [ precedence precedence ][ preference preference ]
        [ brief | truncate ] {
      proto[ all | direct | static | kernel | aggregate |proto ]
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
           restrict ; | [ precedence precedence ]
              [ preference preference ]
        route_filter [ restrict | ( precedence precedence ) |
          (preference preference ) ] ;
    } ;
} ;
generate
     default  |  ( network [ ( mask mask ) | ( ( masklen number
      | / ) number ) ] )
     [ precedence precedence ] [ preference preference ]
      [ noinstall ] {
      proto[ all | direct | static | kernel | aggregate |proto ]
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
          restrict ;|[ precedence precedence ]
           [ preference preference ] {
        route_filter [ restrict | ( precedence precedence ) |
          ( preference preference ) ] ;
    } ;
} ;
```

Aggregation and generation options

default

Specifies that the aggregate route is advertised as a default route, for example, 0.0.0.0/0. When this parameter is used, the aggregate route is not installed as the default route on the local router.

precedence *precedence*

Specifies the precedence to assign to the resulting aggregate route. The default precedence is 130. When the precedence is set for a particular **proto** or **route_filter,** then the precedence is used for comparing against other contributing routes learned from different protocols. In this case, the contributing route with the lowest precedence is ordered first and the aggregate route is assigned that precedence value.

preference *preference*

Specifies the preference to assign to the resulting aggregate route for comparison against other aggregate routes. When the preference is set for a particular **proto** or **route_filter**, then the preference is used for comparing against other contributing routes learned from the same protocol. In this case, the contributing route with the lowest preference is ordered first and the aggregate route is assigned that preference value.

brief

Used to specify that the AS path should be truncated to the longest common AS path. The default is to build an AS path consisting of SETs and SEQUENCEs of all contributing AS paths. The `ATOMIC_AGGREGATE` path attribute is set on truncated AS paths.

noinstall

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 the aggregate route is not installed in the kernel forwarding table when it is active, but it is still eligible to be exported to other protocols.

`truncate`

Used to specify that the AS path should be completely truncated, removing all elements of contributing AS paths. The truncated AS path is used even if there is only one contributing AS path. The `ATOMIC_AGGREGATE` path attribute is set on truncated AS paths.

`proto` *`proto`*

In addition to the special protocols listed, the contributing protocol can be chosen from among any of the ones supported by GRF GateD, which includes BGP, OSPF, and RIP.

`as` *`autonomous_system`*

Restrict selection of routes to those learned from the specified autonomous system.

`tag` *`tag`*

Restrict selection of routes to those with the specified tag.

`aspath` *`aspath_regexp`*

Restrict selection of routes to those that match the specified AS path.

`restrict`

Indicates that these routes are not to be considered as contributors of the specified aggregate. The specified protocol can be any of the protocols supported by GateD.

`route_filter`

See the following section.

A route can only contribute to an aggregate route that is more general than itself. It must match the aggregate under its mask. Any given route can only contribute to one aggregate route, that is the most specific configured, but an aggregate route can contribute to a more general aggregate.

Route filters

Route filters can be used at several levels within the **`aggregate/generate`** statements to further control the acceptance of route advertisements based on the destination address. See “Global filters” on page 3-55 for more information.

GateD State Monitor (GSM)

4

Chapter 4 describes how to use the GSM command.

The GateD State Monitor (GSM) provides an interactive interface to a running GateD daemon that can be used to query internal GateD variables. You can log into it by executing the **`gsm`** command from the CLI or through telnet from a UNIX prompt. After user authentication is successfully accomplished and the GSM interface is established, GSM answers any query sent to it as ASCII commands. The commands include such tasks as displaying the GateD routing table and configuration of the routing protocols.

Note: For this release, the GSM can be configured through the GSM statement in the `/etc/gated.conf` file. By using this statement, you can configure the GSM on or off, plus configure the TCP port number to which the GSM binds, the users who are allowed access to the GSM, and the hosts from which GateD permits telnet connections to the GSM. For backward compatibility, the GSM is on by default.

The following sections give complete information on how to establish a GSM interface, including information on using the CLI or UNIX prompt, the user authentication process, and how to query using the ASCII commands:

Establishing a GSM connection 4-2

Using the GSM help command 4-5

Displaying route tables. 4-7

Frequently-used commands 4-8

Establishing a GSM connection

The GateD State Monitor (GSM) provides an interactive interface from which you can interrogate the state of route tables, interfaces, routing protocols, and other internal parameters by using the GSM query commands. The following sections explain how to establish a GSM interface, and how to query for information.

Note: For backwards compatibility, the GSM is on by default. For more information, see the "GSM statement" section in Chapter 3. Permission level is system.

GSM connection options

There are two methods for establishing a GSM interface: by using the CLI **`gsm`** command or through a direct telnet connection. The following sections describe these two methods.

CLI interface

When you execute **`gsm`** from the CLI prompt, it returns information about the GateD running on that local GRF. To obtain GateD statistics for a different GRF system, you can use the *`hostname`* option to establish GSM connection to that GRF.

```
super> gsm [ hostname ]
```

Note: As shown in the following example, the CLI help function does not provide information on the various GSM query commands, only how to establish a GSM interface.

For descriptions of GSM commands, log onto GSM and enter the following command:

```
super> ? gsm
Usage: gsm [hostname]
super> gsm ?
usage: telnet [-l user] [-a] host-name [port]
super>
```

To establish a GSM interface from the CLI, you need the password for the administrative Netstar ID account; the default password for this account is NetStar. If you change the administrative password, use the new one during the authentication process.

Starting with the 1.4.8 release, access to the GSM can be configured through the `/etc/gated.conf` file. The most noticeable difference is that you may not be allowed to connect to the GSM or, if the connection is allowed, you may be prompted for a user name and a password. See the "GSM command" in Chapter 3 for more information about configuring GSM.

The following example shows how to establish a GSM interface if it is not configured through the `/etc/gated.conf` file:

```
super> gsm
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Password? xxxxxxxx (for example, NetStar)
```

The following example shows how to establish a GSM interface if it has been configured to prompt for a user name through the `/etc/gated.conf` file:

```
super> gsm
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
User?yyyyyyyy (must be a user name as specified in the /etc/gated.conf file)
Password? xxxxxxxx (must be the password for the user entered in the line above)
```

Because the GSM can also be configured to refuse connections from specific hosts, you may receive the following message

```
telnet: Unable to connect to remote host: Connection refused after
the trying 127.0.0.1... .
```

If this happens, check the **`gsm`** statement in `/etc/gated.conf` to see if it excludes the loopback address.

After authentication has successfully completed, the GSM interface starts. The **`gsm`** prompt uses the machine domain name, as shown in the following example:

```
Gated State Monitor. Version GateD R3_5Beta_3; CVS Branch:A1_4_1;
Path:/
/gated/code/GSM
GateD-router.sitename.com>
```

It is at this point that you can use the **`help`** command for information on how to query for information. See the “Using the GSM help command” on page 4-5 for more information.

Telnet connection

From a UNIX shell, you can establish a GSM interface by opening a telnet connection on the TCP port specified in `/etc/gated.conf` or by default, TCP port 616 to the machine running GateD. You can telnet from the administrative LAN or from the GateD machine itself. The syntax for the telnet command is as follows:

`telnet` *`host-name`* `[` *`port`* `]`

`host-name`

Name or IP address of machine running GateD.

`port`

TCP port `616` or configured port to access GSM.

To telnet to the GSM interface, you must use the password from the administrative Netstar ID account; the default password shipped with your system is `NetStar`. If you have changed the administrative password, use the new password.

In this release, access to the GSM can be configured through the `/etc/gated.conf` file; the most noticeable difference is that when you try to connect using the **`telnet`** command, you may not be allowed to connect to the GSM, or if the connection is allowed, you may be prompted for a user name, along with the administrative password.

The following example shows how to use the **`telnet`** command to log into GateD running on a router at address 10.22.22.22 and connecting to TCP port **`616`** (the GSM is not configured through the `/etc/gated.conf` file in this example). Note that the **`gsm`** prompt contains the machine domain name:

```
# telnet 10.22.22.22 616

    Trying 10.22.22.22...

    Connected to 10.22.22.22.

    Escape character is '^]'.

    Password?xxxxxxxx  (for example, NetStar)

    Gated State Monitor. Version GateD R3_5Beta_3;

    Path:

    GateD-router.sitename.com>
```

The following example shows how to establish a GSM interface if it has been configured to prompt for a user name through the `/etc/gated.conf` file:

```
# telnet 10.22.22.22 616

    Trying 10.22.22.22...

    Connected to 10.22.22.22.

    Escape character is '^]'.

    User? yyyyyyyy        (must be a user name as specified in the /etc/gated.conf file)

    Password?xxxxxxxx     (must be the password for the user entered in the line above)

    Gated State Monitor. Version GateD R3_5Beta_3;

    Path:

GateD-router.sitename.com>
```

Because the GSM can be configured to refuse connections from specific hosts (through the `/etc/gated.conf` file), you can receive the following message:

```
    telnet: Unable to connect to remote host: Connection refused after
    trying 10.22.22.22...
```

If this happens, check the **`gsm`** statement in `/etc/gated.conf` to see if it excludes the loopback address.

Once the GSM interface has been successfully established, you can use the **`help`** command for information on how to query for information. The following section describes the **`help`** command.

Using the GSM help command

Once the GSM interface connection has been establish, the GSM answers queries submitted through a series of commands. The top-level **`help | ?`** command provides the list of available commands. Commands may be abbreviated when no confusion is possible (see the Abbreviating commands section for more information). Some of these commands have associated subcommands which are explained in following sections. The following example shows how to use the **`help`** command:

```
GateD-router.sitename.com> help

    HELP: The possible commands are:
    ?     : Print help messages
    help  : Print help messages
    show  : Show internal values
    quit  : Close the session
    enable: Enable the session
    exec  : Execute actions (must be enabled)
    exit  : Close the session
```

Second-level show subcommands

To view information regarding different GRF components, you can use the second-level **`show`** subcommand, as shown in the following example:

```
GateD-router.sitename.com> show

    HELP: The possible show subcommands are:
    version  : Show the current GateD version
    kernel   : Show the Kernel support
    iso      : Show the ISO support
    interface [name|index]: Show interface status
    memory   : Show the memory allocation
    ip       : Show info about IP protocol
    task     : Show list of active tasks
    ospf     : Show info about OSPF protocol
    timer    : Show list of timers
    bgp      : Show info about BGP protocol
    rip      : Show info about rip protocol
    isis     : Show info about ISIS protocol
```

Third-level commands

You can use third-level commands to obtain a finer granularity of information for a specific GRF component, as shown in the following example:

```
GateD-router.sitename.com> show ip
    HELP: The possible subcommands are:
    sum       Show info about IP routes
    exact     [x.x.x/len]: Show info about specific IP routes
    aggregate [x.x.x/len]: Show info about specific Aggregate routes
    all       Show entire FIB
    consume   [x.x.x/len]: Show consuming route and more specific route
    lessspec   Show less specific routes per protocol
    refines    Show more specific routes per protocol
```

The following example shows the use of the **show** command to display OSPF options:

```
GateD-router.sitename.com> show ospf
    HELP: The possible subcommands are:
    summary  : Show OSPF Summary
    errors   : Show OSPF Error Counters
    interface: Show interface status
    io stats : Show I/O stats
    nexthops : Show OSPF nexthops
    ase      : Show OSPF ASE Database
    lsdb     : Show OSPF LSDB Database
    border   : Show OSPF ASB/AB RTR Database
```

The following example shows the use of the **show** command to display BGP options:

```
GateD-router.sitename.com> show bgp
    HELP: The possible subcommands are:
    summary: Show BGP summary
    peeras [AS number]: Show BGP peer info
    group  [AS number]: Show BGP group summary
    aspath : Show BGP aspath info
```

Displaying route tables

To display and view IP route tables through the GSM, establish a GSM session and use the **`show ip all`** command as shown in the following example:

```
GateD-router.sitename.com> sh ip all
Sta        78.78.78/24 203.3.1.153     IGP (Id 1)
Sta        99.99.98/24 202.1.1.153     IGP (Id 1)
Sta        99.99.99/24 212.1.3.152     IGP (Id 1)
Sta             127/8  127.0.0.1       IGP (Id 1)
Sta      198.174.11/24 206.146.160.1   IGP (Id 1)
Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
Dir         203.3.1/24 203.3.1.151     IGP (Id 1)
Dir        204.10.1/24 204.10.1.151    IGP (Id 1)
Sta   204.100.1.147/32 208.1.1.152     IGP (Id 1)
Sta     205.2.4.138/32 212.1.2.134     IGP (Id 1)
Dir     206.146.160/24 206.146.160.151 IGP (Id 1)
Dir         208.1.1/24 208.1.1.151     IGP (Id 1)
Dir         212.1.1/24 212.1.1.151     IGP (Id 1)
Dir         212.1.2/24 212.1.2.151     IGP (Id 1)
Dir         212.1.3/24 212.1.3.151     IGP (Id 1)
```

In the following example, the output for **`show ip cons`** displays different information for a specific route:

```
GateD-router.sitename.com> sh ip cons 202/8
No less specific routes found for IP route 202 mask 255
More specific routes for IP route 202 mask 255...
Dir         202.1.1/24 202.1.1.151     IGP (Id 1)
Dir         202.1.2/24 202.1.2.151     IGP (Id 1)
Dir         202.1.3/24 202.1.3.151     IGP (Id 1)
Dir         202.5.2/24 202.5.2.151     IGP (Id 1)
Dir         202.5.4/24 202.5.4.151     IGP (Id 1)
```

Frequently-used commands

The following commands have proven useful in managing and debugging GateD configurations:

`s ip al`

All routes to be installed in kernel.

`s ip ref` *`[protocol]`* **`0`**

All routes learned through a protocol.

`s ip ref al` *`xx`*

All routes starting with *`xx`*

`s ip exact x.x.x/len`

Everything we know about a route.

For example, the following shows how to display all routes with **`99`** in first octet:

```
GateD-router.sitename.com> s ip re al 99
   OSP         99.82.1/24 10.2.1.82       (666) IGP (Id 3)
   OSP     99.82.82.82/32 10.2.1.82       (666) IGP (Id 3)
   Sta        99.175.1/24 206.146.160.1   IGP (Id 2)
   Sta  99.175.175.175/32 206.146.160.1   IGP (Id 2)
```

The following example shows how to display **`exact`** host route information:

```
GateD-router.sitename.com> s ip ex  99.82.82.82/32
Route 99.82.82.82 Mask 255.255.255.255 Entries 1 Announced 1 Depth 0 <>
                      Instability Histories:
Proto  Route    NextHop Proto-prec/Pref Metric/2   Tag Installed
* OSPF_ASE 99.82.82.82  10.2.1.82   150/-   1/30   C0000000 67:49:48
       Forwarding Interface: 10.2.1.175(gf010)
       Status <Int Ext Active Gateway>
       ASPATH (666) IGP (Id 3)
```

Abbreviating commands

Abbreviating commands is acceptable provided the abbreviation unambiguously identifies an entity, as shown in the following example:

```
GateD-router.sitename.com> sh ip sum
   IP radix tree: 38 nodes, 20 routes\
```

The following example shows the shortest abbreviation of a command that works:

```
   GateD-router.sitename.com> s ip al
```

Another form of abbreviation is shown in the following example, where the value `x.x.x/len` can be replaced with **`0`** to show all:

```
   GateD-router.sitename.com> s ip ref b 0      # show all routes
                                                # learned through BGP
```

Glossary of Terms

5

Terms used in descriptions throughout this document are defined here:

adjacency —Relationship formed between selected neighboring routers for the purpose of exchanging routing information. Not every pair of neighboring routers becomes adjacent.

autonomous system —Set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using an exterior gateway protocol to route packets to other ASs.
Since this classic definition was developed, it has become common for a single AS to use several interior gateway protocols and sometimes several sets of metrics within an AS. The use of the term Autonomous System stresses that even when multiple IGPs and metrics are used, the administration of an AS appears to other ASs to have a single coherent interior routing plan and to present a consistent picture of what networks are reachable through it. The AS is represented by a number between 1 and 65534, assigned by the Internet Assigned Numbers Authority.

BGP, Border Gateway Protocol —Class of exterior gateway protocols, described in more detail in the BGP section of the Protocol Overview.

cost —OSPF metric. See metric.

designated router —Designated router generates a link state advertisement for the multi-access network and assists in running the protocol.

destination —Network or any host.

distance —EGP metric. See metric. Valid values are from zero to 255 inclusive.

egp, exterior gateway protocol, exterior routing protocol —Class of routing protocols used to exchange routing information within an autonomous system. A detailed explanation of exterior gateway protocols is available in the Protocol Overview.

FIB—Table in the kernel that controls the forwarding of packets is a forwarding table, ISO-defined as the forwarding information base.

gateway —Intermediate destination by which packets are delivered to their ultimate destination. A host address of another router that is directly reachable via an attached network. As with any host address it can be specified symbolically.

gateway_list —List of one or more gateways separated by white space.

host —IP address of any host. Usually specified as a dotted quad, four values in the range of 0 to 255 inclusive separated by dots (.). For example `132.236.199.63 or 10.0.0.51.`

It can also be specified as an eight digit hexadecimal string preceded by `0x`. For example `0x????????` or `0x0a000043`. Finally, if `noresolv` is not specified, a symbolic hostname such as `gated.cornell.edu` or `nic.ddn.mil` is usable. The numeric forms are much preferred over the symbolic form.

interface —Host address of an attached interface. This is the address of a *broadcast*, *nbma* or *loopback* interface and the remote address of a point-to-point interface. As with any host address, it can be specified symbolically.

interface —Connection between a router and one of its attached networks.

A physical interface can be specified by a single IP address, domain name, or interface name. unless the network is an unnumbered point-to-point network. Multiple levels of reference in the configuration language allow identification of interfaces using wildcard, interface type name, or delete word address. Be careful with the use of interface names as future Unix operating systems can allow more than one address per interface. Dynamic interfaces can be added or deleted, and indicated as up or down as well as changes to address, netmask and metric parameters.

igp, interior gateway protocol, interior routing protocol —One of a class of routing protocols used to exchange routing information within an autonomous system. A detailed explanation of interior gateway protocols is available in the Protocol Overview.

interface_list —List of 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.

IS-IS —One of a class of interior gateway protocols, not supported in current GRF releases.

local_address —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 can be specified symbolically.

mask —Means of subdividing networks using address modification. A mask is a dotted quad specifying which bits of the destination are significant. Except when used in a route filter, GateD only supports contiguous masks.

mask length —Number of significant bits in the mask.

metric —One of the units used to help a system determine the best route. Metrics can be based on hop count, routing delay, or an arbitrary value set by the administrator depending on the type of routing protocol. Routing metrics can influence the value of assigned internal preferences. (See preference.)

multiaccess networks —Physical networks that support the attachment of multiple (more than two) routers. Each pair of routers on such a network is assumed to be able to communicate directly.

neighbor —Router which with implicit or explicit communication is established by a routing protocol. Neighbors are usually on a shared network, but not always. This term is mostly used in OSPF and EGP. Usually synonymous with peer.

neighboring routers —Two routers that have interfaces to a common network. On multi-access networks, routers are dynamically discovered by the OSPF HELLO protocol.

network —Packet-switched network. A network can be specified by its IP address or network name. The host bits in a network specification must be zero. Default can be used to specify the default network (0.0.0.0).

network —P address of a network. Usually specified as a dotted quad, one to four values, in the range 0 to 255 inclusive, separated by dots (.), for example, `132.236.199, 132.236` or `10`. It can also be specified as a hexadecimal string preceded by `0x` with an even number of digits, of length between two and eight. For example `0xnnnnnnn, 0xnnnn,` or `0x0n`.
Also allowed is the symbolic value `default` that has the distinguished value `0.0.0.0`, the default network. If `noresolv` is not specified, a symbolic network name such as `nr-tech-prod`, `cornellu-net` and `arpanet` is usable. The numeric forms are preferred over the symbolic form.

number —Positive integer.

OSPF, Open Shortest Path First—One of a class of interior gateway protocols.

ospf_area —OSPF allows collections of contiguous networks and hosts to be grouped together. Such a group, together with the routes having interfaces to any of the included networks, is called an area.
Each area runs a separate copy of the basic link-state algorithm. This means that each area has its own topological database. The topology of an area is invisible from the outside of the area. Conversely, routers internal to a given area know nothing of the detailed topology external to the area. This isolation of knowledge enables OSPF to effect a marked reduction in routing traffic as compared to treating the entire autonomous system as a single link-state domain.

peer —Router which with implicit or explicit communication is established by a routing protocol. Peers are usually on a shared network, but not always. This term is mostly usedby BGP. Usually synonymous with neighbor.

port —UDP or TCP port number. Valid values are from 1 through 65535 inclusive.

precedence (protocol-precedence) —Precedence is a value between 0 (zero) and 255, used to set the predecence of routing protocols. The protcol with the best (numerically lowest) precedence contains the prefixes found in the kernel forwarding table and exported to other protocols. A default precedence is assigned to each source from which GateD receives routes (See preference.)

preference —Preference is a value between 0 (zero) and 65536, used to select between many routes to the same destination. The route with the best (numerically lowest) preference is the active route. The active route is the one installed in the kernel forwarding table and exported to other protocols. A default preference is assigned to each source from which GateD receives routes; it has the value of 0. (See precedence.)

prefix —Contiguous mask covering the most significant bits of an address. The prefix length specifies how many bits are covered.

QoS, quality of service —OSI equivalent of TOS.

RIP, Routing Information Protocol —One of a class of interior gateway protocols.

route filter —Configurable entity based on destination address or destination address and mask specified to match a route that is to be filtered out and then ignored when such a route is advertised.

router id —32-bit number assigned to each router running the OSPF protocol. This number uniquely identifies the router within the autonomous system.

router_id —IP address used as unique identifier assigned to represent a specific router. This is usually the address of an attached interface.

RIB, routing information base, routing database, routing table —Repository of all of GateD's retained routing information, used to make decisions and as a source for routing information that is to be propagated. ISO-defined as the routing information base.

simplex —Interface can be marked as simplex either by the kernel, or by interface configuration. A simplex interface is an interface on a broadcast media that is not capable of receiving packets it broadcasts.
GateD takes advantage of interfaces that are capable of receiving their own broadcast packets to monitor whether an interface appears to be functioning properly.

time —Time value, usually a time interval. It can be specified in any one of the following forms:

`number`

Non-negative decimal number of seconds. For example, `27, 60` or `3600`.

`number:number`

A non-negative decimal number of minutes followed by a seconds value in the range of zero to 59 inclusive. For example, `0:27`, `1:00` or `60:00`.

`number:number:number`

A non-negative decimal number of hours followed by a minutes value in the range of zero to 59 inclusive followed by a seconds value in the range of zero to 59 inclusive. For example, `0:00:27`, `0:01:00` or `1:00:00`.

`time to live`
`ttl`

`Time To Live (TTL) of an IP packet. Valid values are from one (1) through 255, inclusive.`

TOS, type of service —Type of Service is for internet service quality selection. The type of service is specified, along the abstract parameters precedence, delay, throughput, reliability, and cost. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram travels. The vast majority of IP traffic today uses the default type of service.

Dynamic Routing RFCs

A

The following is a list of RFCs relevant to dynamic routing:

RFC 827
E. Rosen, *Exterior Gateway Protocol EGP*

RFC 891
D. Mills, *Dcn Local-network Protocols*

RFC 904
D. Mills, *Exterior Gateway Protocol Formal Specification*

RFC 1058
C. Hedrick, *Routing Information Protocol*

RFC 1105
K. Lougheed, Y. Rekhter, *Border Gateway Protocol BGP*

RFC 1163
K. Lougheed, Y. Rekhter, *A Border Gateway Protocol (BGP)*

RFC 1164
J. Honig, D. Katz, M. Mathis, Y. Rekhter, J. Yu,
Application of the Border Gateway Protocol in the Internet

RFC 1227
M. Rose, *SNMP MUX Protocol and MIB*

RFC 1245
J. Moy, *OSPF Protocol Analysis*

RFC 1246
J. Moy, *Experience with the OSPF Protocol*

RFC 1253
F. Baker, R. Coltun, *OSPF Version 2 Management Information Base*

RFC 1256
S. Deering, *ICMP Router Discovery Messages*

RFC 1265
Y. Rekhter, *BGP Protocol Analysis*

RFC 1266
Y. Rekhter, *Experience with the BGP Protocol*

RFC 1267
K. Lougheed, Y. Rekhter, *A Border Gateway Protocol 3 (BGP-3)*

RFC 1268
P. Gross, Y. Rekhter, *Application of the Border Gateway Protocol in the Internet*

RFC 1269
J. Burruss, S. Willis,
Definitions of Managed Objects for the Border Gateway Protocol (v. 3)

RFC 1321
R. Rivest, *The MD-5 Message Digest Algorithm*

RFC 1370
Internet Architecture Board, *Applicability Statement for OSPF*

RFC 1388
G. Malkin, *RIP Version 2 Carrying Additional Information*

RFC 1397
D. Haskin, *Default Route Advertisement In BGP2 And BGP3 Versions Of The Border Gateway Protocol*

RFC 1403
K. Varadhan, *BGP OSPF Interaction*

RFC 1583
J. Moy, *OSPF Version 2*

GateD Configuration File

B

Appendix B contains a list of the statements and keywords allowed in the GateD configuration file, `/etc/gated.conf`.

There is no template to edit for the GateD configuration file. To create `/etc/gated.conf`, you open a new file in the `/etc` directory and name the file “`gated.conf`.” Then, you enter each statement needed for your site configuration.

A statement is made up of keywords and variables. A semi-colon separates one statement from another. Some statements contain a single statement, look at the Option statement near the end of this page. Most statements are themselves made up of a series of statements. The Trace options statement below has two statements. Keywords and variables are defined and described in Chapter 3.

Syntax description conventions

In this manual, keywords and special characters that the parser expects exactly as shown appear in **`bold`** monospaced font. Variable parameters are shown in *`italic`* monospaced font. Square brackets ([and]) are used to show optional keywords and parameters. A vertical bar (|) separates optional parameters. Parentheses ( ) group keywords and parameters, when necessary.

For example, in the following syntax example, the square brackets indicate that the parameters are optional. The keywords are **`backbone`** and **`area`**. The vertical bar indicates that either **`backbone`** or **`area`** *`area`* can be specified. Because *`area`* is in italic font, it is a variable parameter provided by you. The parentheses group a keyword and its variable parameter.

[**backbone** | (**area** *area*)]

In this manual, keywords and special characters that the parser expects exactly as shown appear in **`bold`** monospace font. Variable parameters are shown in *`italic`* monospace font. Square brackets ([ ]) show optional keywords and parameters. A vertical bar (|) separates optional parameters. Parentheses ( ) group keywords and parameters when necessary.

The GateD configuration file, `/etc/gated.conf`, consists of a series of statements. A semicolon (`;`) terminates a segment and separates statements.

Statements are composed of tokens separated by white space that can be any combination of blanks, tabs, and new lines. This structure identifies the parts of the configuration associated with each other and with specific protocols.

Comments can be specified in either of two forms:

– beginning with a pound sign (#) and running to the end of the line

– using C style, starting with a `/*` and continuing until it reaches `*/`

Order of statements

Entering a statement out of order causes an error when `/etc/gated.conf` is parsed. The type of configuration statements and the order in which these statements appear in `/etc/gated.conf` are as follows:

– options statements

– GSM statements

– interface statements

– definition statements

– protocol statements

– static statements

– control statements

– aggregate statements

Two types of statements do not fit in these categories: %directive statements and %trace statements. These statements provide instructions to the parser and control tracing from `/etc/`

`gated.conf`. They do not relate to the configuration of any protocol and can appear anywhere in the `/etc/gated.conf` file.

The statements appear in the order in which they should be entered into the file.

# Trace options statement

```
traceoptions [ trace_file [ replace ] [ size size [ k|m ] files files ] ]
        [ control_options ] trace_options [ except trace_options ] ;
traceoptions none ;
```

# Options statement

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

# GSM statement

```
gsm  ( yes  |  no  |  on  |  off )
[ {
      [ port port-number ; ]
      [ usernames user-list ; ]
      [ hosts  host-set ; ]
}  ]  ;
```

# Interface statement

```
interfaces {
      options
           [ strictinterfaces ]
           [ scaninterval time_seconds ]
           ;
      interface interface_list
           [ precedence precedence ]
           [ down precedence precedence ]
           [ passive ]
           [ simplex ]
           [ reject ]
           [ blackhole ]
```

```
                    ;
               define address
                    [ broadcast address ] | [ pointtopoint address ]
                    [ netmask mask ]
                    [ multicast ]
                    ;
          } ;
```

# Definition statement

```
          autonomoussystem autonomous_system [ loops number ] ;
          multipath (  yes  |  no  |  off  |  no )  ;
          routerid host ;
          confederation confederation ;
          routing-domain rdi ;
          martians {
                    host host [ allow ] ;
                    network [ allow ] ;
                    network mask mask [ allow ] ;
                    network ( masklen | / ) number [ allow ] ;
                    default [ allow ] ;
          } ;
```

# RIP statement

```
          rip  ( yes | no | on | off  ) [ {
               broadcast ;
               nobroadcast ;
               nocheckzero ;
               precedence precedence ;
               defaultmetric metric ;
               query authentication [none |[ simple | md5 ] password)] ;
               interface interface_list
                     [ noripin ] | [ ripin ]
                     [ noripout ] | [ ripout ]
                     [ metricin metric ]
                     [ metricout metric ]
                        [ version 1 ]|[ version 2 [ multicast|broadcast ]]
                      [ [ secondary ] authentication [ none |
                     ( [ simple | md5 ] password ) ] ] ;
```

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

# OSPF statement

```
      ospf  ( yes | no | on | off  ) [ {
              defaults {
                     precedence precedence ;
                     cost cost ;
                     tag [ as ] tag ;
                     type 1 | 2 ;
                     inherit-metric ;
            } ;
             exportlimit routes ;
             exportinterval time_seconds ;
             traceoptions trace_options ;
             monitorauthkey authkey ;
             monitorauth none | ( [ simple | md5 ] authkey ) ;
             backbone | ( area area ) {
                   authtype  none | simple ;
                   stub [ cost cost ] ;
                  networks {
                       network [ restrict ] ;
                       network mask mask [ restrict ] ;
                       network ( masklen | / ) number [ restrict ] ;
                       host host [ restrict ] ;
                  } ;
                  stubhosts {
                        host  cost cost ;
                  } ;
                  interface interface_list ; [ cost cost ] {
                         interface_parameters
                  } ;
                   interface interface_list nonbroadcast [ cost cost ] {
                        pollinterval time_seconds ;
                        routers {
                              gateway [ eligible ] ;
                        } ;
```

```
                    interface_parameters
                } ;
                # Backbone only:
                  virtuallink neighborid router_id  transitarea area {
                       interface_parameters
                 } ;
             } ;
        } ] ;
```

# BGP statement

```
bgp  ( yes | no | on | off )
{
     [ precedence precedence ; ]
     [ preference preference ; ]
     [ allow bad community ; ]
     [ defaultmetric metric ; ]
     [ traceoptions trace_options ; ]
     [ clusterid host ; ]
     [ disable export best ; ]
[ group type
     ( external peeras autonomous_system
          [ ignorefirstashop ]  [ subgroup integer ]
          [ metricout metric ] [ med ]  [ nexthopself ]
          [ send-no-community | send-community ]
          { bgp_peer_stmts } ;

   | ( internal peeras autonomous_system
           [ reflector-client [ no-client-reflect ] ]
           [ ignorefirstashop ]
           [ lcladdr local_address ]
           [ outdelay time ]
           [ metricout metric ]
           [ subgroup integer ]
           [ nexthopself ]
           [ send-no-community | send-community ]
           { bgp_peer_stmts } ;

   | ( routing peeras autonomous_system proto proto_list
             interface interface_list
           [ reflector-client [ no-client-reflect ] ]
```

```
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ subgroup integer ]
                [ nexthopself ]
                [ send-no-community | send-community ]
                { bgp_peer_stmts } ;

        | ( confed peeras autonomous_system proto proto_list
                  interface interface_list
                [ reflector-client [ no-client-reflect ] ]
                [ ignorefirstashop ]
                [ lcladdr local_address ]
                [ outdelay time ]
                [ metricout metric ]
                [ reflector-client [ no-client-reflect ] ]
                [ subgroup integer ]
                [ nexthopself ]
                [ send-no-community | send-community ]
                { bgp_peer_stmts } ;

        | ( test peeras autonomous_system ) ]
            {
                [ allow {
                  network
                  network mask mask
                  network ( masklen | / ) number
                  all
                  host host ]
             } ;
                  peer host
                [ metricout metric ]
                [ ignorefirstashop ]
                [ nogendefault ]
                [ gateway gateway ]
                [ precedence precedence ]
                [ preference preference ]
                [ lcladdr local_address ]
```

```
                    [ holdtime time ]
                    [ version number ]
                    [ passive ]
                    [ sendbuffer number ]
                    [ recvbuffer number ]
                    [ outdelay time ]
                    [ keep [ all | none ] ]
                    [ show-warnings ]
                    [ noaggregatorid ]
                    [ keepalivesalways ]
                    [ v3asloopokay ]
                    [ nov4asloop ]
                    [ ascount count ]
                    [ throttle count ]
                    [ allow bad routerid ]
                    [ logupdown ]
                    [ ttl ttl ]
                    [ traceoptions trace_options ]
                    [ nexthopself ]
                    ;
                } ;
            } ;
              } ;
```

# Weighted route dampening statement

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

# ICMP statement

```
icmp {
    traceoptions trace_options ;
}
```

# Router discovery server statement

```
                routerdiscovery server  ( yes | no | on | off ) [ {
                       traceoptions trace_options ;
                       interface interface_list
                              [ minadvinterval time_seconds ] |
                              [ maxadvinterval time_seconds ] |
                              [ lifetime time ]
                              ;
                       address interface_list
                              [ advertise ] | [ ignore ] |
                              [ broadcast ] | [ multicast ] |
                              [ ineligible ] | [ preference preference ]
                              ;
                } ] ;
```

# Router discovery client statement

```
                routerdiscovery client  ( yes | no | on | off ) [ {
                    traceoptions trace_options ;
                    precedence precedence ;
                    interface interface_list
                           [ enable ] | [ disable ]
                           [ multicast ] | [ broadcast ]
                           [ quiet ] | [ solicit ]
                           ;
                } ] ;
```

# Kernel statement

```
                kernel {
                      options
                             [nochange]
                             [noflushatexit ]
                             ;
                       routes number ;
                       flash
                             [ limit number ]
                          [ type interface | interior | all ]
                          ;
                       background
                            [ limit number ]
                            [ priority flash | higher | lower ]
                ;
            traceoptions trace_options ;
        } ;
```

# Static statement

```
        static {
            ( host host ) | default |
             ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
                gateway gateway_list
                [ interface interface_list ]
                [ precedence precedence ]
                [ retain ]
                [ reject ]
                [ blackhole ]
                [ noinstall ] ;
             ( network [ ( mask mask ) | ( ( masklen | / ) number ) ] )
                interface interface
                [ precedence precedence ]
                [ retain ]
                [ reject ]
                [ blackhole ]
                [ noinstall ] ;
        } ;
```

# Filtering statement

```
        network mask mask [ exact|refines|between number and number ]
        network masklen |/ number [ exact | refines |
        between number and number ]
        all default host host
```

# Autonomous system matching statement

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

# AS path attribute statement for importing routes

```
        aspath-opt {
            [community autonomous_system : community-id | community-id ]
            [ community no-export| no-advertise | no-export-subconfed| none ]
        }
```

# AS path attribute statements for exporting routes

```
aspath-opt {
    [community autonomous_system : community-id | community-id ]
    [ community no-export| no-advertise | no-export-subconfed| none ]
    [community any : community-id ]
    [community autonomous_system : any ]
    [community any : any ]
}

mod-aspath {
    [community (( autonomous_system : community-id )
      |community-id | no-export | no-advertise | no-export-subconfed )]
    [delete community (( autonomous_system : community-id )
      |community-id | no-export | no-advertise | no-export-subconfed )]
    [delete community autonomous_system : any ]
    [delete community any : community-id ]
    [delete community any : any ]
}
```

# Importing from BGP statement

```
import proto bgp as autonomous_system
    [ subgroup integer ] [ aspath-opt] restrict ;
import proto bgp as autonomous_system
    [ subgroup integer ] [ aspath-opt ]
    [ precedence precedence ][ preference preference ]
     [ localpref preference ] {
    route_filter  [ restrict | ( precedence precedence )
    | ( preference preference ) | ( localpref preference ) ] ;
} ;


import proto bgp aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ aspath-opt ] restrict ;
import proto bgp aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ aspath-opt ] [ precedence precedence ] [ preference
      preference ] [ localpref preference ]
     route_filter [ restrict | ( precedence precedence )
    | ( preference preference ) | ( localpref preference  } ;
```

# Importing from RIP and Redirect statement

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

# Importing from OSPF statement

```
import proto ospfase [ tag ospf_tag ] restrict ;
import proto ospfase [ tag ospf_tag ]
    [ precedence precedence ] {
    route_filter  [ restrict | ( precedence  precedence ) ;
} ;
```

Export statement

```
        restrict
        [ all ] metric [ igp ] metric
        localpref preference
```

Exporting to BGP statement

```
export proto bgp as autonomous system
      restrict ;
export proto bgp as autonomous system [ mod-aspath ]
     [ subgroup integer ] [ metric metric ]|
      [ localpref preference] {
      export_list ;
} ;
```

Exporting to RIP statement

```
        export proto rip
             [ ( interface interface_list ) | ( gateway gateway_list ) ]
             restrict ;
        export proto rip
             [ ( interface interface_list ) | ( gateway gateway_list ) ]
             [ metric metric ] {
        export_list ;
        } ;
```

Exporting to OSPF statement

```
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
     restrict ;
export proto ospfase [ type 1 | 2 ] [ tag ospf_tag ]
     [ metric metric ] {
     export_list ;
} ;
```

Note: Keep the order of the OSPF export statements exactly as shown; reversing them causes a parser error.

# Exporting BGP routes

```
proto bgp as autonomous_system   [ subgroup integer ]
     restrict ;
proto bgp as autonomous_system  [ subgroup integer ]
     [ metric metric  | localpref preference ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

# Exporting RIP routes

```
proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     restrict ;
proto rip
     [ ( interface interface_list ) | ( gateway gateway_list ) ]
     [ metric metric ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

# Exporting OSPF routes

```
proto ospf |ospfase restrict ;
proto ospf | ospfase [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

# Exporting routes, non-routing by interface

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

# Exporting routes, non-routing by protocol

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

# Exporting by AS path

```
proto bgp | all aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     restrict ;
proto bgp | all aspath aspath_regexp
     origin any | ( [ igp ] [ egp ] [ incomplete ] )
     [ metric metric  | localpref preference ] {
     route_filter [ restrict | ( metric metric ) ] ;
} ;
```

# Exporting by route tag

```
proto rip | ospf | all tag tag restrict ;
proto rip | ospf | alltag tag
    [ metric metric ] {
    route_filter [ restrict | ( metric metric ) ] ;
} ;
```

# Aggregation and generation statements

```
aggregate
     default  |  ( network [ ( mask mask ) | ( ( masklen
      number | / ) number ) ] )
      [ precedence precedence ][ preference preference ]
        [ brief | truncate ] {
      proto[ all | direct | static | kernel | aggregate |proto ]
           [ ( as autonomous_system ) | ( tag tag )
               | ( aspath aspath_regexp ) ]
           restrict ; | [ precedence precedence ]
              [ preference preference ]
        route_filter [ restrict | ( precedence precedence ) |
          (preference preference ) ] ;
    } ;
} ;
```

```
            generate
                 default  |  ( network [ ( mask mask ) | ( ( masklen number
                  | / ) number ) ] )
                 [ precedence precedence ] [ preference preference ]
                  [ noinstall ] {
                  proto[ all | direct | static | kernel | aggregate |proto ]
                       [ ( as autonomous_system ) | ( tag tag )
                           | ( aspath aspath_regexp ) ]
                      restrict ;|[ precedence precedence ]
                       [ preference preference ] {
                    route_filter [ restrict | ( precedence precedence ) |
                      ( preference preference ) ] ] ;
                }  ;
            }  ;
```

# Directive statement

```
%directory directory
```

# Include statement

```
%include filename
```

Configuration files for A

C

This chapter contains the complete configuration files for Configuration A, which is explained in Chapter 2, “GateD Configuration Tutorial.”

/etc/grifconfig.conf file

The following is the complete `/etc/grifconfig.conf` for configuration A.

```
#NetStar $Id: grifconfig.conf,v 1.10 1997/08/01 17:24:04 pargal Exp $
#
# Configuration file for GigaRouter/GRF interfaces.
#
# The contents of this file specify the IP addressing information for
# the networks attached to the system’s interfaces.  This includes
# interfaces on media cards as well as directly attached interfaces
# such as de0 or ef0 (maintenance Ethernet) or lo0 (software loopback).
#
# The addresses of directly attached interfaces are configured
# directly from this file by /etc/netstart calling the grifconfig(8)
# script.
#
# The addresses of the interface(s) on a given media card are
# configured into the BSD/OS kernel when the media card boots and
# comes on line.
#
# Each entry in this file has the following format:
#
# name  address               netmask        broad_dest     arguments
#
# The name of a GigaRouter interface encodes the hardware type,
# GigaRouter cage number, slot number, and interface number.
#
#   --  The first character must be ‘g’ (to specify a GigaRouter
#       interface).
#   --  The second character is the hardware type of the
#       interface:  ‘a’ for ATM, ‘e’ for ETHERNET,
#       ‘f’ for FDDI, ‘h’ for HIPPI, ‘p’ for PPP, ‘s’ for HSSI.
#       (‘l’ is also used, for GigaRouter software loopback.)
#   --  The third character is the number of the GigaRouter cage.
#       (Currently this must be ‘0’, as multiple GigaRouter cages
        #       are not yet supported.)
#   --  The fourth character is the hex digit (0 through f) of
#       the slot number within the GigaRouter cage.
#   --  The fifth (and sixth) characters specify the number of the
#       LOGICAL interface on the card:
#
#               For ATM cards, the fifth and sixth characters are
#               the hex digits of the logical interface.  Logical
#               interfaces numbered from 0 to 7f are on the top
#               physical connector on the ATM card, and logical
#               interfaces numbered 80 to ff are on the bottom
#               physical connector.  NOTE:  These logical interface
#               numbers are NOT the same as the VPI/VCI numbers
#               of a PVC (see /etc/grpvc.conf for that).
#
#               For FDDI cards, the fifth character will be 0, 1,
#               2, or 3 to specify the logical interface on the
#               FDDI card.  NOTE:  The logical interface number
#               may be different from the physical interface on
#               the card, depending on the single- or dual-
#               attachedness of the various interfaces.  Examples:
#               “gf073” specifies the bottom-most connector
#               on the FDDI card in slot 7; “gf020” specifies
#               top-most connector on the FDDI card in slot 2,
#               or the top TWO connectors on that card if they’re
#               configured dual-attached.
#
#               For ETHERNET cards, the fifth character will be 0, 1,
#               2, 3, 4, 5, 6 or 7 to specify the physical interface
#               on the ETHERNET card.
#               Examples:
#               “ge067” specifies the 8th physical connector
#                       on the ETHERNET card in slot 6.
#               "ge000" specifies the first (top) connector on the
#                       ETHERNET card in slot 0.
#               "ge0f7" specifies the last (bottom) connector on the
#                       ETHERNET card in slot 15.
#
#               For HIPPI cards, which only have one interface,
#               the fifth character is always 0.  Example:
#               "gh0f0" specifies the interface for a HIPPI card
#               in slot 15.
#
# The IP "address", "netmask" (optional), and "broad_dest" (optional)
# address fields must be specified in canonical IP dotted-quad nota-
tion.
# An entry of "-" (a single hyphen) may be specified for any of these
# fields as a place-holder.  This may be useful, e.g., if no netmask
# is desired but a broadcast or destination address must be specified
# in the next field.
#
#
# For this release, the "broad_dest" field specifies the broadcast
# IP address for Ethernet & FDDI interfaces, and the destination of
# a point-to-point ATM or HIPPI interface.
#
# The "arguments" field is for any additional arguments to be supplied
# to the underlying ifconfig(8) command that will be executed by
# grifconfig(8).  The most useful purpose would be to specify an
# MTU value for the interface using the "mtu" keyword of ifconfig(8).
# The keyword "iso" can also be specified here which designates the
current
# line as an iso address entry.
# See the example entry below, and the man page for ifconfig(8).
#
#
# NOTE: All interface names are case sensitive !
# Always use lower case letters when defining interface names.
#
#
# name  address         netmask         broad_dest      arguments
#
```

```
de0     206.146.165.2   255.255.255.0     #  A: Control board ethernet interface.
lo0     127.0.0.1       255.0.0.0         # A: Standard loopback interface.
lo0     10.254.254.11   255.255.255.255  # A: Loopback for RID (always active).
ga010   10.200.1.11     255.255.255.0     #  A: ATM interface to AS 10001.
```

/etc/gratm.conf file

The following is the complete `/etc/gratm.conf` file for configuration A.

```
#
# NetStar $Id: gratm.conf,v 1.7 1996/09/19 19:58:10 wdp Exp $
#
# gratm.conf - GigaRouter ATM Configuration File
#

#
# This file is used to configure GigaRouter ATM interfaces.
# Statements in this file are used to configure ATM PVC’s,
# signalling protocols, arp services, and traffic shapes.
#
# gratm(8) uses this file as input when it is run by grinchd(8)
# whenever an ATM media card boots to configure the card.
#

#
# gratm.conf is divided into five sections:
#
# The Service section is where ATM ARP services are defined (entries
# defined in this section are referenced from the Interface section
# of this file to define which ARP service an interface should use).
#
# The Traffic Shaping section is where traffic shapes are defined
# (entries defined in this section are referenced from the Interface
# and PVC sections of this file to define which traffic shapes
# interfaces and PVC’s should use).
#
# The Signalling section is where the signalling protocol to be used
#       by a physical interface to establish Switched Virtual Circuits
#       is specified.
#
```

```
# The Interfaces section is where per-logical-interface parameters
#       such as ARP services and Traffic shapes are bound to specific
#       logical interfaces.
#
# The PVC section is where Permanent Virtual Circuits are defined,
#       using traffic shapes defined in the Traffic Shaping section,
#       along with other parameters specific to PVC configuration.
#


#
# Notes on the format of this file:
#
# Comments follow the Bourne Shell style (all characters following a #
# on a line are ignored).
#
# Statements in this file are separated by newlines.  A statement may
# span multiple lines by ending each incomplete line of the statement
# with a ‘\’ character.  Example:
#
# Traffic_Shape name=high_speed peak=15000      # this is a statement
#
# Service name=bc0 type=bcast addr=198.174.11.1 \
#       addr=198.176.11.1            # this is also a statement
#
# User-defined names for Traffic_Shape’s and Service’s must be defined
# before they are used, ie., the definition of a traffic shape must
# precede its use in an Interface or PVC specification, and the
# definition of a Service must precede the use of the name of that
# service in defining any Interfaces.
#


#
# ARP Service info
#
# Lines beginning with the keyword “Service” define virtual “services”
which
# may or may not be present on an ATM network attached to a GigaRouter.
#
# Each Service entry the ATM configuration file has the following for-
mat:
```

```
#
# Service name=value    type=arp|bcast  addr=value [addr=value ...]
#
# The “name” field is a unique name to identify this ATM service
#
# The “type” field specifies the type of ATM service being configured.
# and how the address argument(s) which follow are interpreted.
#
#       type=arp        Indicates an ARP service.  One to three “addr”
#                      fields must follow, defining NSAP addresses of
#                       ARP services on the attached ATM network.
#
#                       A logical interface using this “Service” entry
#                      will connect to one of the addresses defined for
#                      this service to get ATM address information for
#                       any IP addresses it does not already know the
#                       ATM address of.
#
#    type=bcast    Defines a “broadcast service”.  The “addr” field(s)
#                  contain IP addresses of hosts on a given logical
#                 logical ATM network to which copies of any broadcast
#                  packets will be sent, allowing the GigaRouter, when
#                  so configured, to simulate broadcast over a logical
#                 IP network.
#

#Service name=arp0 type=arp
addr=47000580ffe1000000f21513eb0020481513eb00


#
# Traffic shaping parameters
#
# Lines beginning with the keyword “Traffic_Shape” define
# traffic shapes which may be used to configure the performance
# characteristics of ATM Virtual Circuits.
#
# The Traffic_Shape’s defined here are to be referenced by name when
# to assign traffic shapes to PVC’s or Interfaces later in this
# configuration file.  (See Examples in the PVC or Interface section
# of this file for examples on how to reference traffic shapes defined
here.)
#
# Each Traffic_Shape entry the ATM configuration file has the format:
#
# Traffic_Shape name=value peak=bps [sustain=bps burst=cells]
# [qos=high|low]
#
# The “name” field is a unique name to identify this ATM service, so we
# can refer to collection of peak, [sustain, burst], [qos] parameters
# as a group when configuring PVC’s or Interfaces later in this file.
#
# The ‘peak’, ‘sustain’, and ‘burst’ fields specify, respectively,
# the peak cell rate, the sustained cell rate, and the burst rate.
# The values for ‘peak’ and ‘sustain’ are in kilobits per second (max
# of 155000), and the value for ‘burst’ is in cells (maximum of 2048).
#
# The ‘qos’ (Quality of Service) field specifies which rate queues
# to use.  A value of ‘high’ corresponds to high priority service
# which uses high-priority queues, and a value of ‘low’ corresponds
# to low priority service which uses the low-priority rate queues.
#
# The peak rate is the only parameter which is mandatory.  If ommitted,
# the sustain and burst rates are set to match the peak rate.  If qos
# is not specified, it defaults to “high”.
#

#Traffic_Shape name=medium_speed_low_quality \
#       peak=75000 qos=low

#Traffic_Shape name=low_speed_high_quality \
#       peak=15000 qos=high

#
# Signalling parameters
#
# Lines beginning with the keyword “Signalling” define
# the signalling protocol which will be used on a physical
# ATM interface to establish Switched Virtual Circuits for
# any logical interfaces on the named physical interface.
```

```
#
# Physical interfaces on GigaRouter ATM cards are identified by
# the slot number of the interface card in the GigaRouter chassis
# in hex notation (0-f) plus the location of the physical interface
# on the card (either the top connector, or the bottom connector on the
card).
#
# Each Signalling entry the ATM configuration file has the format:
#
# Signalling card=hex connector=top|bottom \
#   [protocol=UNI3.0|UNI3.1|NONE]  [mode=SDH|SONET]
#
# The ‘card’ and ‘connector’ specification are mandatory.
#
# The card should be identified by a hexidecimal digit representing
# the slot number of the card in the GigaRouter chassis.
#
# The connector should be either ‘top’ or ‘bottom’.
#
# The ‘protocol’ parameter defines the signalling protocol to be used
# in the setup of Switched Virtual Circuits (SVC’s) on this physical
# interface.  This parameter is optional.  If left unspecified, the
# ATM card uses UNI3.0 signalling by default.
#
# Valid values for the signalling parameter include:
#
#          UNI3.0    - for the UNI 3.0 signalling protocol.
#
#          UNI3.1    - for the UNI 3.1 signalling protocol.
#
#          NONE      - for no signalling protocol.
#
# The ‘mode’ specification is optional. It can be either SDH or SONET.
# By default, it uses SONET.
#

#Signalling card=a connector=top protocol=UNI3.1
#Signalling card=a connector=bottom protocol=NONE


#
```

```
# Interfaces
#
# Lines beginning with the keyword “Interface” define
# GigaRouter logical ATM interfaces.
#
# The format of a logical interface definition is:
#
# Interface ifname [service=service_name] [traffic_shape=shape_name]
#
# The optional ‘service’ parameter allows an ATM service to be
# be defined for this logical interface (the ‘service_name’ must be
# a name defined in the Services section above).
#
# The optional ‘traffic_shape’ parameter allows a traffic shape
# to be defined for this logical interface (the ‘shape_name’ must be
# a named defined as a Traffic_Shape above).
#
# If no traffic shape is specified, a default shape of 155Kbps, high
# quality of service is used.
#


#
# PVC’s
#
# Lines beginning with the keyword “PVC” define
# Permanent virtual circuits.
#
# The format of a PVC definition is:
#
# PVC ifname VPI/VCI proto=ip|raw|vc|ipnllc [input_aal=3|5|NONE] \
#       [dest_if=logical_if [dest_vc=VPI/VCI]] [traffic_shape=shape]
#
# The first 3 parameters (ifname, VPI/VCI, and proto) are mandatory.
#
# ‘ifname’ specifies the GigaRouter ATM logical interface in the usual
# format (e.g., ga030, ga0e80).
#
# ‘VPI/VCI’ specifies the (decimal) Virtual Path Identifier and
# Virtual Circuit Identifier of the PVC, separated by a slash (/).
```

```
#
# ‘proto’ specifies the protocol to be supported on this PVC.  Legal
# values are ‘ip’ for Internet Protocol (with LLC/SNAP headers), ‘vc’
# for VC Based Multiplexing of IP (as specified in RFC 1483), and ‘raw’
# for raw adaptation layer (AAL-5 or AAL-3/4) packets.
#
# If ‘proto’ specified is ‘raw’, the ‘dest_if’ parameter specifies the
# GigaRouter interface of the destination for this raw adaptation layer
# connection (specified in the same GigaRouter interface format
# described above), and (optionally) the dest_vc parameter specifies
# the destination VPI/VCI.
#
# ‘input_aal’ parameter may be used to specify the adaptation layer,
#       input_aal=3     specifies AAL-3/4
#       input_aal=5     specifies AAL-5
#
# The optional ‘traffic_shape’ parameter allows a traffic shape
# to be defined for this logical interface (the ‘shape_name’ must be
# a named defined as a Traffic_Shape above).
#
# If no traffic shape is specified, a default shape of 155Kbps, high
# quality of service is used.
#

# ATM traffic shape:
# peak=peak cell rate, sustain=sustained cell rate, burst=burst size
# Each peak rate specified, automatically creates a rate queue.
#
Traffic_shape name=high_speed_high_quality peak=155000 \
sustain=155000  burst=2048 qos=high   # A: Use entire OC-3 bandwidth.

# Signalling:
# The default setting for protocol is NONE and is not needed when
# configuring a PVC.
Signalling card=1 connector=top protocol=NONE   # A: Signalling is not
required for PVCs.

# Interface:
# Interface ifname [service=service_name] [traffic_shape=shape_name]
# service and traffic_shape are optional
```

```
Interface ga010   # A: To AS 10001.


# PVC:
# PVC is where you map your logical interface to the VPI/VCI pair
# and specify traffic shaping.
PVC ga010  0/33 proto=ip traffic_shape=high_speed_high_quality
```

/etc/gated.conf file

fThe following is the complete `/etc/gated.conf` for configuration A.

```
interfaces {
  interface all passive;   #A: Maintain same preference regardless of
interface status.
};


routerid 10.254.254.11;    #A: RID for GRF1
autonomoussystem 10000;


rip no;   #A: RIP is off by default in 1.3.[7-8]?


bgp on {
  traceoptions “/var/tmp/gated_bgp”
  replace size 200k files 2 all;


  group type external peeras 10001 {   #A: Group type external means
EBGP
    peer 10.200.1.12;
  };
};


import proto bgp as 10001 { all; };   #A: Importing all BGP routes from
10001


export proto bgp as 10001 {   #A: Exporting all AS 10000 routes into
                              #BGP toward 10001
  proto bgp as 10000 { all; };
};
```

GSM displays for configuration A

The following is the complete GateD State Monitor (GSM) displays for configuration A.

```
GateD-GRF1.customer.com> show bgp summary

Neighbor       V AS    Est.# Est(s) #routes #active #to MsgRx Msg State

10.200.1.12    4 10001 6    9     16      16       8      3      9
Established

BGP summary, 1 groups, 1 peers


GateD-GRF1.customer.com> show bgp detail prx 10001 10.200.1.12 0/0

group type External AS 10001 local 10000 Flags <>

  Peer: 10.200.1.12 ID: 10.200.1.11 Version: 4  Gateway: (null)

  Peer is reachable through interface: 10.200.1.11  (ga010)

  Local ID: 10.254.254.11

  Local Addr:

                10.200.1.11+179

  Local port 179  remote port 2086

  Flags 0x820

  State 0x6

  Established Transitions 5 Established Time 1298

  LastSt: OpenConfirm LastEv: RecvKeepAlive LastErr: None

  Options 0x0 <>

  MED exported -1

  Proto-precedence/BGP preference 170/-

  Number of Notifications from this peer: 0, Number of Notifications
sent to peer 0

  Number of routes from this peer: 16 Number of active routes from this
peer: 16

  Number of routes exported to this peer: 8

  Messages in 30 (updates 2, not updates 28) 700 octets

  Messages out 32 (updates 4, not updates 28) 739 octets

  Last traffic (seconds): Received 46, Sent 4, Checked 21

  Time since last Keepalive sent: 4 seconds

  Time since last Update Recv’d: 1298 seconds

  Max Update Tx per second: unlimited

  Inbound Timer: 0

  Outbound Timer: 0

  Received and buffered Octets: 0

  Active Holdtime: 180

  Route Queue Timer:
    unset
  Route Queue: empty


 Define symbols: * = active route + best BGP route
                 + = active route -- not BGP route
                 ^ = best BGP route (but not active)
                 - = not best BGP route (not active)
                 @ = suppressed BGP route
                 | = not BGP, or not avail. for selection
    Route           Mask           NextHop          Prec/Pref Metric/
2     Tag ASPath
 *  10.3.2       255.255.255     10.200.1.12       170/-   -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.3       255.255.255    10.200.1.12        170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.4       255.255.255    10.200.1.12       170/-     -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.5       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.6       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.7       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.8       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.3.9       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 1111 2222 3333 IGP (Id 22) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.2       255.255.255     10.200.1.12       170/-    -1/100    0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.3       255.255.255     10.200.1.12       170/-   -1/100     0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.4       255.255.255     10.200.1.12       170/-   -1/100     0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
```

```
 *  10.4.5      255.255.255    10.200.1.12       170/-    -1/100    0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.6      255.255.255    10.200.1.12       170/-   -1/100     0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.7      255.255.255    10.200.1.12       170/-    -1/100    0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.8      255.255.255    10.200.1.12       170/-    -1/100    0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
 *  10.4.9      255.255.255   10.200.1.12        170/-    -1/100    0
(10000) 10001 4444 5555 6666 IGP (Id 23) orig NH 10.200.1.12
ID 10.200.1.11
```

Index

%directory
  directive statement, 3-8
%include
  directive statement, 3-8

A

ack
  OSPF statement
    tracing options, 3-27
address
  router discovery server statement, 3-44
advertise
  router discovery server statement, 3-44
aggregate statement, 3-75
all
  BGP statement
    tracing options, 3-39
  route filtering, 3-54
allow
  BGP statement, 3-35
allow bad community
  BGP statement, 3-31
allow bad routerid
  BGP statement, 3-38
any
  AS path
    aspath-opt, 3-63
argument field, in grifconfig.conf file, 2-5
as
  aggregate statement, 3-78
AS path
  attributes for communities, 3-61
  for export, 3-62
  for import, 3-62
  mod-aspath, 3-62
  operators, 3-58
  regular expressions, 3-57
  terms, 3-57
AS paths, matching, 3-57
ascount
  BGP statement, 3-38
aspath
  aggregate statement, 3-78
aspath-opt
  route filters, 3-66
assigning protocol-precedence, 3-5
authentication
  RIP statement, 3-21
authkey
  OSPF statement, 3-26
authtype
  OSPF statement, 3-24
autonomous system
  BGP connection, 1-9
  description, 1-8
  establishing policy, 1-11
  transit AS, 1-11
autonomoussystem
  definition statement, 3-17

B

backbone area
  OSPF statement, 3-24
background
  kernel statement, 3-49
between number and number
  route filtering, 3-54
BGP
  AS path, 1-10
  confederations, 1-14
  group type, 1-10
  MEDs, 1-17
  path evaluation, 1-11
  policy role, 1-11
  route reflection, 1-19
  route selection criteria, 1-16
  sessions, 1-10
BGP group types, 3-32
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
  routing peeras, 3-32
BGP route selection algorithm, 3-4
BGP statement, 3-28

blackhole
  interface statement, 3-13
  static statement, 3-52
brief
  aggregate statement, 3-77
broadcast
  interface statement, 3-13
    ip addresses/routes, 3-15
  RIP statement, 3-19, 3-21
  router discovery client statement, 3-46
  router discovery server statement, 3-44
broadcast address
  in grifconfig.conf, 2-5

C

clusterid
  BGP statement, 3-31
community
  AS path
    aspath-opt, 3-63
  in BGP, 1-15
community no-advertise
  AS path
    aspath-opt, 3-63
community no-export
  AS path
    aspath-opt, 3-63
community no-export-subconfed
  AS path
    aspath-opt, 3-63
community none
  AS path
    aspath-opt, 3-63
confed peeras, 3-33
confederation
  definition statement, 3-18
confederations, in BGP, 1-14
control statements, 3-53
control_options
  trace statement
    traceoptions, 3-7
controlling exportation of routes
  export statement, 3-68
cost
  OSPF statement, 3-23, 3-26

D

dampen-flap
  weighted route dampening statement, 3-40
dd
  OSPF statement
    tracing options, 3-27
default
  aggregate statement, 3-76, 3-77
  route filtering, 3-54
defaultmetric
  BGP statement, 3-31
  RIP statement, 3-20
defaults
  OSPF statement, 3-23
define
  aggregate list
    route filtering
      global filters, 3-56
  export filter
    route filtering
      global filters, 3-55
  export list
    route filtering
      global filters, 3-56
  filter
    route filtering
      global filters, 3-56
  import filter
    route filtering
      global filters, 3-55
  import statement, 3-64
  interface statement, 3-13
  route filtering
    global filters, 3-55
definition statement, 3-17
delete community
  AS path, 3-63
  no-export
    AS path, 3-63
  no-export-subconfed
    AS path, 3-63
desination, specifying
  export statement, 3-69
destination address
  in grifconfig.conf, 2-5
detail
  BGP statement
    tracing options, 3-39
  trace statement
    packet tracing, 3-7
directive statement, 3-8
disable
  OSPF statement, 3-25
  router discovery client statement, 3-46

disable export best
  BGP statement, 3-31
down precedence
  interface statement, 3-13

E

enable
  OSPF statement, 3-25
  router discovery client statement, 3-46
error
  ICMP statement
    tracing options, 3-42
exact
  route filtering, 3-54
except
  trace statement
    traceoptions, 3-7
export statement, 3-68
  AS path, 3-74
  exporting by protocols
    kernel, 3-73
  exporting routes
    direct, 3-73
    OSPF, 3-73
    RIP, 3-72
    static, 3-73
  non-routing by interface, 3-73
    aggregate, 3-74
    default, 3-74
  non-routing by protocol, 3-74
  route tag, 3-74
exporting routes, 3-72
  to OSPF, 3-71
  to RIP, 3-71
exportinterval
  OSPF statement, 3-24
exportlimit
  OSPF statement, 3-24
external peeras, 3-32
  BGP statement
    group type, 3-32

F

flash
  kernel statement, 3-48
forwarding tables, 3-47

G

ga0yz interface name, 2-4
GateD
  BGP confederations, 1-14
  communities, 1-15
  GateD State Monitor (GSM), 1-14
  MEDs, 1-14
  route preference biasing, 1-16
  route reflection, 1-15
  routing arbiter interaction, 1-16
GateD State Monitor
  commands, 4-1
  features
    GateD
      GateD State Monitor, 4-1
GateD syntax, 3-2, B-2
gateway
  BGP statement
    peer, 3-36
  options statement, 3-9
gdc reconfig command, 3-39
gendefault
  options statement, 3-9
generate statement, 3-75
global filters, 3-55
group clauses
  BGP statement, 3-31
group type, BGP, 1-10
gsm statement, 3-9
GSM, GateD State Monitor, 1-14, 4-1

H

hello
  OSPF statement
    tracing options, 3-27
hellointerval
  OSPF statement, 3-26
holdtime
  BGP statement, 3-37
host
  route filtering, 3-54
hosts
  gsm statement, 3-10

I

ICMP statement, 3-41
  tracing options
    packets, 3-41

iflist
  kernel statement
    tracing options, 3-49
ignore
  router discovery server statement, 3-44
ignorefirstashop
  BGP statement, 3-34, 3-36
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
  routing peeras, 3-32
import statement, 3-64
  from OSPF, 3-67
  redirect, 3-66
  routes from BGP, 3-65
importing routes
  from BGP, 3-66
  from OSPF, 3-67
  from RIP/redirect, 3-66
ineligible
  router discovery server statement, 3-45
info
  ICMP statement
    tracing options, 3-42
  kernel statement
    tracing options, 3-50
inherit-metric
  OSPF statement, 3-24
interface
  BGP statement, 3-33
  confed peeras, 3-33
  interface statement, 3-13
  kernel statement
    tracing options, 3-50
  OSPF statement, 3-25, 3-26
  RIP statement, 3-20
  router discovery client statement, 3-45
  router discovery server statement, 3-44
  routing peeras, 3-32
  static statement, 3-52
interface lists
  interface statement, 3-14
interface statement, 3-12
interfaces section, gratm.conf file, 2-6
internal peeras, 3-33
IP address
  in grifconfig.conf, 2-4
IP interface addresses and routes, 3-15

K

keepalive
  BGP statement
    tracing options, 3-39
keepalivesalways
  BGP statement, 3-38
keep-history
  weighted route dampening statement, 3-41
kernel statement, 3-47
  background
    higher, 3-49
  tracing options, 3-49
    remants, 3-49

L

lcladdr
  BGP statement, 3-34, 3-36
  internal peeras, 3-33
  routing peeras, 3-32
lcladdrL
  confed peeras, 3-33
lifetime
  router discovery server statement, 3-44
localas
  BGP statement, 3-36
localpref
  export statement, 3-70
  import statement, 3-64
  route filters, 3-66
logupdown
  BGP statement, 3-38
loopback
  interface statement
    ip addresses/routes, 3-15
lsabuild
  OSPF statement
    tracing options, 3-27
lsuLOSPF statement
  tracing options, 3-27

M

mark
  options statement, 3-9
martians
  definition statement, 3-18
matching AS paths, 3-57
maxadvinterval
  router discovery server statement, 3-44

max-flap
  weighted route dampening statement, 3-40
med
  BGP statement, 3-34
  external peeras, 3-32
MEDs, 1-14, 1-17
  originating in BGP, 1-17
metric
  export statement
    controlling exportation of routes, 3-68
metricin
  RIP statement, 3-20
metricout
  BGP statement, 3-36
  confed peeras, 3-33
  internal peeras, 3-33
  RIP statement, 3-21
  routing peeras, 3-32
metrout
  BGP statement, 3-34
minadvinterval
  router discovery server statement, 3-44
mod-aspath
  AS path, 3-62
  export statement, 3-70
monitorauthkey
  OSPF statement, 3-24
multicast
  interface statement, 3-14
  RIP statement, 3-21
  router discovery client statement, 3-46
  router discovery server statement, 3-44
multipath
  definition statement, 3-17

N

nbma
  interface statement
    ip addresses/routes, 3-15
netmask
  in /etc/grifconfig.conf, 2-5
  interface statement, 3-14
networks
  OSPF statement, 3-25
nexthopself
  BGP statement, 3-34, 3-39
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
  routing peeras, 3-32
noaggregatorid
  BGP statement, 3-37
nobroadcast
  RIP statement, 3-19
nochange
  kernel statement, 3-48
nocheckzero
  RIP statement, 3-20
no-client-reflect
  confed peeras, 3-33
  internal peeras, 3-33
  routing peeras, 3-32
noflushatexit
  kernel statement, 3-48
nogendefault
  BGP statement, 3-36
noinstall
  aggregate statement, 3-77
  static statement, 3-52
nomulticast
  OSPF statement, 3-26
nonbroadcast
  OSPF statement, 3-26
non-broadcast multi-access
  interface statement
    ip addresses/routes, 3-15
none
  trace statement, traceoptions, 3-7
noresolv
  options statement, 3-9
noripin
  RIP statement
    interface, 3-20
noripout
  RIP statement, 3-20
normal
  trace statement, traceoptions, 3-7
nosend
  options statement, 3-9
nov4asloop
  BGP statement, 3-38

O

open
  BGP statement
    tracing options, 3-39
options
  interface statement, 3-12
  kernel statement, 3-48
options statement, 3-8
OSPF
  BGP interaction, 1-12
OSPF statement, 3-22

other
  kernel statement
    tracing options, 3-50
  RIP statement
    tracing options, 3-22
outdelay
  BGP statement, 3-34, 3-37
  confed peeras, 3-33
  internal peeras, 3-33
  routing peeras, 3-32

P

packet tracing
  trace statement, 3-7
packets
  BGP statement
    tracing options, 3-39
  RIP statement
    tracing options, 3-21
passive
  BGP statement, 3-37
  interface statement, 3-13
  OSPF statement, 3-25
path selection, BGP, 1-11
peer
  BGP statement, 3-35
  parameters
    BGP statement, 3-35
  subclauses
    BGP statement, 3-31
point-to-point
  interface statement, 3-15
pointtopoint
  interface statement, 3-14
policy
  BGP role, 1-11
pollinterval
  OSPF statement, 3-26
port
  gsm statement, 3-10
precedence
  aggregate statement, 3-77
  BGP statement, 3-30, 3-36
  import statement, 3-64
  interface statement, 3-13
  options statement, 3-9
  OSPF statement, 3-23
  RIP statement, 3-20
  router discovery client statement, 3-45
  static statement, 3-52
preference
  aggregate statement, 3-77
  BGP statement, 3-30, 3-36
  import statement, 3-64
  router discovery server statement, 3-45
priority
  OSPF statement, 3-25
priority flash
  kernel statement, 3-49
proto
  aggregate statement, 3-78
  BGP statement, 3-34
  confed peeras, 3-33
  routing peeras, 3-32
protocol-precedence
  assigning, 3-5
protocol-precedence and preference, 3-4
protocol-precedence and preference, GateD, 1-30
PVC section, gratm.conf file, 2-7

Q

query authentication
  RIP statement, 3-20
quiet
  router discovery client statement, 3-46

R

reach-decay
  weighted route dampening statement, 3-41
recovering interfaces, 3-39
recv
  BGP statement
    tracing options, 3-39
  trace statement
    packet tracing, 3-7
recvbuffer
  BGP statement, 3-37
redirect
  ICMP statement
    tracing options, 3-41
  kernel statement
    tracing options, 3-50
refines
  route filtering, 3-54
reflector-client
  BGP statement, 3-34
  confed peeras, 3-33
  internal peeras, 3-33
  routing peeras, 3-32

reject
  interface statement, 3-13
  static statement, 3-52
remnants
  kernel statement
    tracing options, 3-49
replace
  trace statement
    traceoptions, 3-6
request
  OSPF statement
    tracing options, 3-27
  RIP statement
    tracing options, 3-21
response
  RIP statement
    tracing options, 3-22
restrict
  aggregate statement, 3-78
  export statement
    controlling exportation of routes, 3-68
  route filters, 3-66
retain
  static statement, 3-52
retransmitinterval
  OSPF statement, 3-25
reuse-below
  weighted route dampening statement, 3-40
RIP statement, 3-19
RIP, in GateD, 1-27
ripin
  RIP statement, 3-20
ripout
  RIP statement, 3-20
route advertisements, BGP, 1-10
route aggregation, 3-75
route filtering, 3-53
  exporting, 3-69
route filters, 3-65
  in aggregation/generation, 3-78
route generation statement, 3-75
route preference biasing, 1-16
route reflection, 1-15
  configuring, 1-19
route server, and GateD, 1-16
route_filter
  aggregate statement, 3-78
router discovery
  client, 3-43
  client statement, 3-45
  protocol, 3-42
  server statement, 3-43
routerdeadinterval
  OSPF statement, 3-26
routerdiscovery
  ICMP statement
    tracing options, 3-42
routerid
  definition statement, 3-18
routers
  OSPF statement, 3-26
routes
  kernel statement, 3-48
    tracing options, 3-50
routing arbiter interaction, 1-16
routing peeras, 3-32
routing tables, 3-47
routing-domain
  definition statement, 3-18

S

scaninterval
  interface statement, 3-13
send
  BGP statement
    tracing options, 3-39
  trace statement
    packet tracing, 3-7
sendbuffer
  BGP statement, 3-37
send-community
  BGP statement, 3-35
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
  routing peeras, 3-32
send-no-community
  BGP statement, 3-35
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
  routing peeras, 3-32
service section, gratm.conf file, 2-5
sessions, BGP, 1-10
show-warnings
  BGP statement, 3-37
signalling section, gratm.conf file, 2-6
simplex
  interface statement, 3-13
size
  trace statement
    traceoptions, 3-6
solicit
  router discovery client statement, 3-46

sourcegateways
  RIP statement, 3-21
spf
  OSPF statement
    tracing options, 3-27
state
  router discovery client, 3-46
static statement, 3-51
strictinterfaces
  interface statement
    options, 3-13
stub
  OSPF statement, 3-25
stubhosts
  OSPF statement, 3-25
subgroup
  BGP statement, 3-34
  confed peeras, 3-33
  external peeras, 3-32
  internal peeras, 3-33
suppress-above
  weighted route dampening statement, 3-40
symbols
  kernel statement
    tracing options, 3-49
syslog upto
  options statement, 3-9

T

tag
  aggregate statement, 3-78
  export statement
    OSPF, 3-72
  OSPF statement, 3-23
throttle
  BGP statement, 3-38
trace statement, 3-6
trace_file
  trace statement
    traceoptions, 3-6
traceoptions
  BGP statement, 3-31, 3-39
  OSPF statement, 3-24
  RIP statement
    interface, 3-21
  router discovery client statement, 3-45
  router discovery server statement, 3-44
traceoptions statement, 3-6
  ICMP statement, 3-41
tracing options
  BGP statement, 3-39
traffic shaping section, gratm.conf file, 2-6
transitdelay
  OSPF statement, 3-25
truncate
  aggregate statement, 3-78
trustedgateways
  RIP statement, 3-21
ttl
  BGP statement, 3-39
type
  OSPF statement, 3-24
  routes
    kernel statement, 3-49
type 1, type 2
  exporting routes
    to OSPF, 3-72

U

unreachable messages, BGP, 1-10
unreach-decay
  weighted route dampening statement, 3-41
update
  BGP statement
    tracing options, 3-39
usernames
  gsm statement, 3-10

V

v3asloopokay
  BGP statement, 3-38
version
  BGP statement
    peer parameters, 3-37
version 1
  RIP statement, 3-21
version 2
  RIP statement, 3-21
virtuallink neighborid
  OSPF statement, 3-27
virtuallink transitarea
  OSPF statement, 3-27

W

weighted route dampening
  in GateD, 1-32
weighted route dampening statement, 3-40