VYATTA, INC. | Vyatta OFR

Vyatta OFR Configuration Guide

Vyatta
Suite 160
One Waters Park Drive
San Mateo, CA 94403
vyatta.com

ISSUE DATE: May 2007

DOCUMENT RELEASE NO. 2.1.1

DOCUMENT REVISION NO. 2.1 v02.

DOCUMENT PART NO. A0-0085-10-02

Table of Contents

Quick List of Examples

Use this list to help you locate examples you'd like to try or look at.

Preface

This guide explains how to use the Vyatta system router, and how to use Vyatta system router commands in the command-line interface. It provides an overview of the router's functionality, highlighting core concepts, and a detailed description of each available command.

This preface provides information about using this guide. The following topics are covered:

- Intended Audience
- Organization of This Guide
- Document Conventions
- Vyatta Publications

Intended Audience

This guide is intended for experienced system and network administrators. Depending on the functionality to be used, readers should have specific knowledge in the following areas:

- Networking and data communications
- TCP/IP protocols
- General router configuration
- Routing protocols
- Network administration
- Network security

Organization of This Guide

This guide has the following aids to help you find the information you are looking for:

- **Quick List of Examples**

  Use this list to help you locate examples you'd like to try or look at.

- **Quick Guide to Configuration Statements**
- Use this section to quickly see the complete syntax of configuration statements.

This guide has the following chapters and appendixes:

Chapter	Description	Page
Chapter 1: System Management	This chapter explains how to configure basic system information for the router such as host name, DNS information, and date and time.	1
Chapter 2: Ethernet and VLAN Interfaces	This chapter explains how to configure Ethernet interfaces, the loopback interface, and VLAN interfaces.	17
Chapter 3: Serial Interfaces	This chapter describes how to configure serial interfaces on the Vyatta system.	28
Chapter 4: Tunnel Interfaces	This chapter explains how to configure GRE and IP-in-IP tunnel interfaces.	33
Chapter 4: Basic Services	This chapter explains how to configure system services such as DHCP, HTTP, Telnet, and SSH.	34

Chapter 5: Forwarding and Routing	This chapter provides a brief overview of basic routing topics: traffic fowarding, unicast routing, multicast routing, and multicast topology discovery.	47
Chapter 6: Routing Policies	This chapter describes the policy framework, which you can use to apply policies that influence routing behavior.	58
Chapter 7: Static Routes	This chapter describes how to configure static routes on the Vyatta system.	85
Chapter 8: RIP	This chapter describes how to configure the Routing Information Protocol on the Vyatta system.	89
Chapter 9: OSPF	This chapter describes how to configure the OSPF routing protocol on the router.	89
Chapter 10: BGP	This chapter describes how to configure the Border Gateway Protocol on the Vyatta system.	106
Chapter 11: VRRP	This chapter describes how to configure the Virtual Router Redundancy Protocol on the Vyatta system.	170
Chapter 12: NAT	This chapter describes how to configure NAT on the Vyatta system.	177
Chapter 13: Firewall	This chapter describes how to configure the Firewall on the Vyatta system.	186
Chapter 14: IPsec VPN	This chapter describes how to configure IPsec VPN on the Vyatta router.	196
Chapter 15: User Authentication	This chapter describes how to set up user accounts and user authentication.	260
Chapter 16: Logging	This chapter explains how the Vyatta system generates and manages system logging messages, and describes how to configure logging.	265
Chapter 17: SNMP	This chapter describes how to configure Simple Network Management Protocol on the Vyatta system.	270
Chapter 18: Software Upgrades	This chapter describes how to use the Vyatta system's package mechanism to update your software.	277

Document Conventions

This guide contains advisory paragraphs and uses typographic conventions.

Advisory Paragraphs

This guide may use the following advisory paragraphs:

Warnings alert you to situations that may pose a threat to personal safety, as in the following example:

WARNING *Risk of injury. Switch off power at the main breaker before attempting to connect the remote cable to the service power at the utility box.*

Cautions alert you to situations that might cause harm to your system or damage to equipment, or that may affect service, as in the following example:

CAUTION *Risk of loss of service. Restarting a running router will interrupt service.*

Notes provide information you might need to avoid problems or configuration errors:

NOTE *You must create and configure network interfaces before enabling them for routing protocols.*

Tip: *Use tips to save time and effort.*

Tips (see left) provide helpful information for doing something in a faster or easier way, or for optimizing the performance of your system.

Typographic Conventions

In addition to advisory paragraphs, this document may use the following typographic conventions:

`Courier`	Courier font is used in command syntax sections and in special example paragraphs.
`boldface Courier`	Boldface Courier font is used to show something you enter at a command line.
boldface	Boldface font is used to represent commands or keywords inside a paragraph of ordinary text.
italics	Italic font is used to show arguments and variables, where you supply the value.

<key>	Angle brackets are used to indicate a key on your keyboard. Combinations of keys are joined by plus signs (“+”). An example is **<Ctrl>+<Alt>+<Del>**.
[*arg1*|*arg2*]	Square brackets enclose enumerated options for completing a syntax. The options are separated by a vertical bar. An example is **[enable|disable]**.
num1–numN	The typographic convention at left indicates a range of numbers. An example is 1–65535, which means 1 through 65535 inclusive.
arg1..argN	The typographic convention at left indicates a range of enumerated values. An example is **eth0..eth23**, which means **eth1**, **eth2**, **eth3**, and so on through **eth23**.
arg [*arg* …]	The typographic convention at left indicates a value that can optionally represent a space-separated list of the same kind of element (for example, a space-separated list of IP addresses).

Vyatta Publications

The Vyatta technical library includes the following publications:

Vyatta OFR Quick Start Guide	Explains how to install the router software, and provides some basic configuration to get you started.
Vyatta OFR Configuration Guide	Explains router functions, and steps through sample configurations for every function.
Vyatta OFR Command Reference	Provides a complete description of each command in the CLI.

Chapter 1: System Management

This chapter explains how to configure basic system information for the router such as host name, DNS information, and date and time.

The following topics are covered:

- System Configuration Overview
- Configuring Host Information
- Configuring DNS
- Configuring Date and Time
- Monitoring System Information

System Configuration Overview

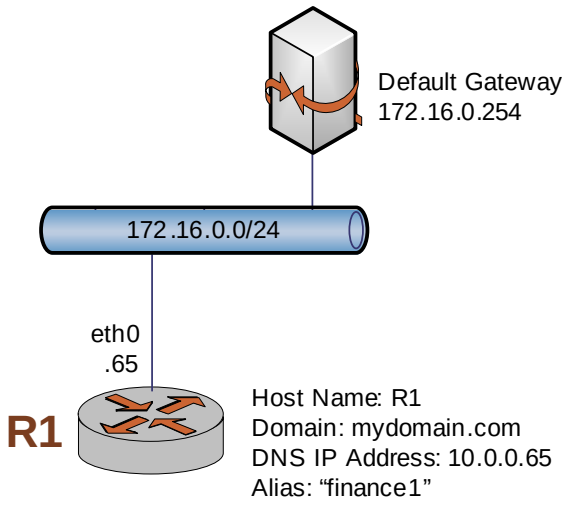
The commands in this chapter allow you to change and view basic IP system information. You can configure the following:

- Host and basic network information
- DNS information
- Date and time information

Configuring Host Information

In this section, sample configurations are presented for the router's host information. The sample configuration used is shown in Figure 1-1.

Figure 1-1 Host information



This section includes the following examples:

- Example 1-1 Setting the router's host name
- Example 1-2 Setting the router's domain
- Example 1-3 Mapping the router's IP address to its host name

- Example 1-4 Setting the default gateway
- Example 1-5 Creating an alias for the router

Host Name

The router's name is set using the **system host-name** command. Router names can include letters, numbers, and hyphens ("-").

Example 1-1 sets the router's host name to R1. To set the router host name, perform the following steps in configuration mode:

Example 1-1 Setting the router's host name

Step	Command

Set the router's host name.	`vyatta@vyatta# set system host-name R1`
	`OK`
	`[edit]`

Commit the change. The command prompt changes to reflect the change	`vyatta@vyatta# commit`
	`OK`
	`[edit]`
	`vyatta@R1>`

Show the configuration.	`vyatta@vyatta# show system host-name`
	`    host-name: "R1"`
	
	`[edit]`

Domain

The router's domain is set using the **system domain-name** command. Domain names can include letters, numbers, hyphens, and periods.

Example 1-2 sets the router's domain to **mydomain.com**.

To set the router's domain, perform the following steps in configuration mode:

Example 1-2 Setting the router's domain

Step	Command

Set the domain name.	`vyatta@R1# set system domain-name mydomain.com`
	`OK`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system domain-name`
	`  domain-name: "mydomain.com"`
	
	`[edit]`

IP Address

The router's IP address can be statically mapped to its host name for local DNS purposes, using the **system static-host-mapping** command.

IP networks are specified in CIDR format—that is, in *ip-address*/*prefix* notation such as 192.168.12.0/24. For single addresses, use dotted quad format, that is, *a.b.c.d*. For network prefixes, enter a decimal number from 1 through 32.

A good practice is to map the router's host name to the loopback address, as the loopback interface is the most reliable on the router. In this example, the loopback interface is given the address 10.0.0.65. This is the address configured for the loopback interface in the sample topology used in this guide.

Example 1-3 creates a static mapping between the router's host name R1 and IP address 10.0.0.65. This is the IP address the DNS server will use to resolve DNS requests for **R1.mydomain.com**.

To map the host name to the IP address, perform the following steps in configuration mode:

Example 1-3 Mapping the router's IP address to its host name

Step	Command

Create a mapping between the host name and the IP address of the loopback interface.	`vyatta@R1# set system static-host-mapping host-name R1 inet 10.0.0.65`
	`OK`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system static-host-mapping`
	`host-name R1 {`
	`inet: 10.0.0.65`
	`}`
	
	`[edit]`

Default Gateway

Example 1-4 specifies a default gateway for the router at 172.16.0.254.

To specify the default gateway, perform the following steps in configuration mode:

Example 1-4 Setting the default gateway

Step	Command

Specify the default gateway.	`vyatta@R1# set system gateway-address 172.16.0.254`
	`OK`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system gateway-address`
	`gateway-address: 172.16.0.254`
	
	`[edit]`

Aliases

You can define one or more aliases for the router by mapping the router's IP address to more than one host name.

Example 1-5 creates the alias **finance1** for the router.

To create an alias for the router, perform the following steps in configuration mode:

Example 1-5 Creating an alias for the router

Step	Command

Define an alias.	`vyatta@R1# set system static-host-mapping host-name R1 alias finance1`
	`OK`
	`[edit]`

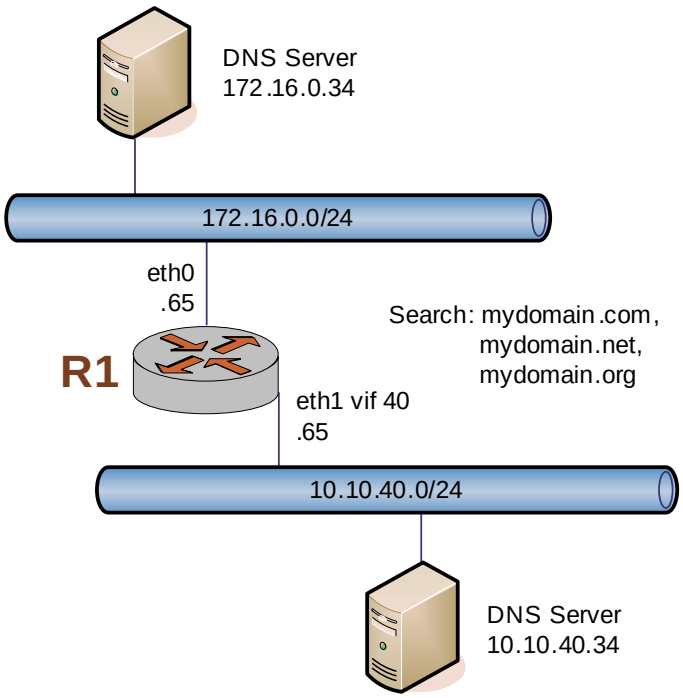
Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system static-host-mapping`
	`    host-name R1 {`
	`        inet: 10.0.0.65`
	`        alias finance1`
	`    }`
	
	`[edit]`

Configuring DNS

In this section, sample configurations are presented for DNS information. The DNS configuration used is shown in Figure 1-2.

Figure 1-2 DNS



This section includes the following examples:

- Example 1-6 Specifying DNS name servers
- Example 1-7 Setting search order for domain completion

DNS Name Servers

DNS name servers are specified using the **system name-server** command.

Example 1-6 specifies two DNS servers for the router: one at 172.16.0.34, and the other at 10.10.40.34.

To specify DNS servers, perform the following steps in configuration mode:

Example 1-6 Specifying DNS name servers

Step	Command
Specify the first DNS server.	`vyatta@R1# set system name-server 172.16.0.34` `[edit]`
Specify the second DNS server.	`vyatta@R1# set system name-server 10.10.40.34` `[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show configuration. (Output is abbreviated here.)	`vyatta@R1# show system`
	`  host-name: "R1"`
	`  domain-name: "mydomain.com"`
	`  name-server 172.16.0.34`
	`  name-server 10.10.40.34`
	`  :`
	`  :`
	`[edit]`

Domain Search Order

You can specify a list of domains for the router to use to complete an unqualified host name. To define this list specify the order in which domains are searched, use the **system domain-search** command.

This command requires a space-separated list of domain names, specified in the order you want them searched. A domain name can include letters, numbers, hyphens (“-”), and periods (“.”).

Example 1-7 directs the router to attempt domain completion in the following order: first, mydomain.com; second, mydomain.net; and last mydomain.org.

To specify domain search order, perform the following steps in configuration mode:

Example 1-7 Setting search order for domain completion

Step	Command

Specify the search order.	`vyatta@R1# set system domain-search mydomain.com mydomain.net mydomain.org`
	`OK`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

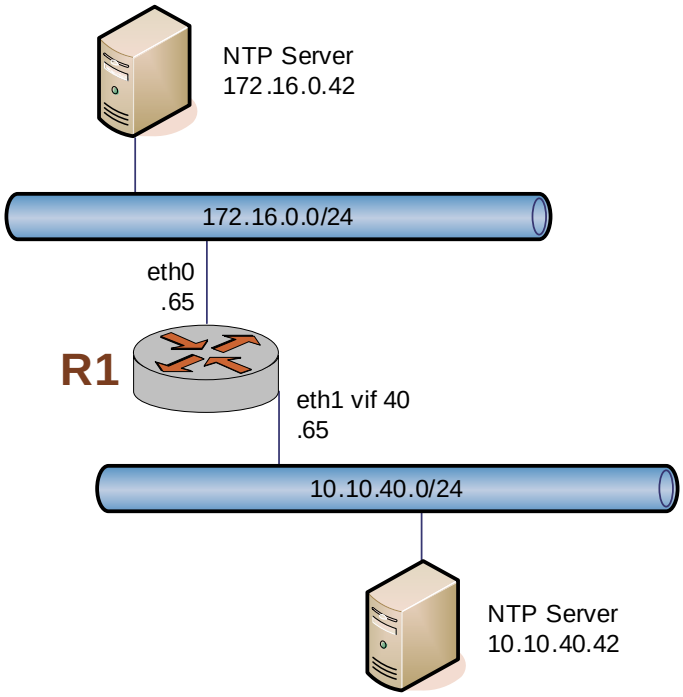
Show the configuration.	`vyatta@R1# show system domain-search`
	`    domain "mydomain.com"`
	`    domain "mydomain.net"`
	`    domain "mydomain.org"`
	
	`[edit]`

Configuring Date and Time

Date and time can either be set manually, or obtained by manually or automatically synchronizing the router with one or more Network Time Protocol (NTP) servers. Time zone must be manually set, and may be specified as an offset from Universal Coordinated Time (UTC) or as one of a number of supported literal time zones.

In this section, sample configurations are presented for maintaining date and time information. The sample configuration used is shown in Figure 1-3.

Figure 1-3 Date and Time



This section includes the following examples:

- Example 1-8 Setting the date and time manually
- Example 1-9 Manually synchronizing the router with an NTP server
- Example 1-10 Setting the time zone as an offset from UTC
- Example 1-11 Setting the time zone as a time zone name
- Example 1-12 Using NTP for automatic synchronization

Setting the Date

Example 1-8 manually sets the date to 1:15 PM exactly on April 24, 2007. The format is *MMDDHHMMYYYY*, and must be enclosed in double quotes.

To manually set the date, perform the following steps in operational mode:

Example 1-8 Setting the date and time manually

Step	Command

Specify the date. The format is *MMDDHHMMYYYY*, enclosed in double quotes.	`vyatta@R1> date “043243152007”`
	`Tue Apr 24 13:15:00 UTC 2007`
	`vyatta@R1>`

Manually Synchronizing with an NTP Server

Example 1-9 manually synchronizes the system clock with the NTP server at 172.16.0.42.

Note that this merely performs a one-time synchronization. It does not set up an ongoing association with the NTP server. For information about setting up automatic synchronization, please see “Using NTP for Automatic Synchronization” on page 13.

To perform a one-time synchronization with an NTP server, perform the following steps in operational mode:

Example 1-9 Manually synchronizing the router with an NTP server

Step	Command

Specify the location of the NTP server.	`vyatta@R1> date ntp 172.16.0.42`
	`Tue Apr 24 13:15:00 UTC 2007`
	`vyatta@R1>`

Setting the Time Zone

Time zone must be configured, using **system time-zone** command. To do this, you specify the amount by which your time zone is offset from UTC (coordinated universal time). UTC has the same time as the Greenwich time zone. The string giving the offset is enclosed in quotes.

The offset you specify is added to UTC to produce the local time. You can also use one of the support time zone name to indicate time zone. Again, the string supplying the time zone name must be enclosed in quotes.

Note that the router uses POSIX-style offsets. The POSIX specification uses positive signs west of Greenwich—not positive signs east of Greenwich, which some other systems use. For example, an offset of “**GMT +4**” corresponds to 4 hours behind UTC (that is, west of Greenwich).

Example 1-10 sets the time zone to 8 hours west of Greenwich, which is Pacific Standard Time.

To set the time zone using an offset from UTC, perform the following steps in configuration mode:

Example 1-10 Setting the time zone as an offset from UTC

Step	Command

Set the time zone.	`vyatta@R1# set system time-zone “GMT+8”`
	`OK`
	`[edit]`
	`vyatta@R1#`

Commit the information.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system time-zone`
	`    time-zone: "GMT+8"`
	
	`[edit]`

The following time zone names, enclosed in double quotes, are also accepted:

“Los Angeles”: Sets the time zone to Los Angeles time.

“New York”: Sets the time zone to New York time.

“Denver”: Sets the time zone to Denver time.

“Chicago”: Sets the time zone to Chicago time.

“Anchorage”: Sets the time zone to Anchorage time.

“Honolulu”: Sets the time zone to Honolulu time.

“Phoenix”: Sets the time zone to Phoenix time.

The default is “**GMT**”, which uses UTC time exactly.

Example 1-11 sets the time zone to Los Angeles time.

To set the time zone using a supported time zone name, perform the following steps in configuration mode:

Example 1-11 Setting the time zone as a time zone name

Step	Command

Specify the time zone as a time zone name.	`vyatta@R1# set system time-zone “Los Angeles”`
	`OK`
	`[edit]`
	`vyatta@R1#`

Commit the information.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Show the configuration.	`vyatta@R1# show system time-zone`
	`    time-zone: "Los Angeles"`
	
	`[edit]`

Using NTP for Automatic Synchronization

To use NTP for automatic synchronization, you must create associations with the NTP servers. To create an association with an NTP server, use the **system ntp-server** command and specify the IP address of the server.

Example 1-12 configures two NTP servers: one at 172.16.0.42, and one at 10.10.40.42.

To specify NTP servers, perform the following steps in configuration mode:

Example 1-12 Using NTP for automatic synchronization

Step	Command
Specify a server at 176.16.0.42.	`vyatta@R1# set system ntp-server 172.16.0.42` `[edit]`
Specify a server at 10.10.40.42.	`vyatta@R1# set system ntp-server 10.10.40.42` `[edit]`

Commit the information.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Example 1-12 Using NTP for automatic synchronization

Show the configuration. (Output is abbreviated here.)

```
vyatta@R1# show system
   host-name: "R1"
   domain-name: "mydomain.com"
   domain-search {
      domain "mydomain.com"
      domain "mydomain.net"
      domain "mydomain.org"
   }
   name-server 172.16.0.34
   name-server 10.10.40.34
   time-zone: "Los Angeles"
   ntp-server "172.16.0.42"
   ntp-server "10.10.40.42"
   :
   :
[edit]
```

Monitoring System Information

This section presents the following topic:

- System Operational Commands

System Operational Commands

You can use the following operational commands to monitor system information. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`clear arp`	Clears the ARP cache.
`init-floppy`	Formats a floppy diskette and prepares it to receive a configuration file.
`mount`	Mounts the floppy disk file system.
`reboot`	Reboots the router.
`show arp`	Displays the ARP cache.
`show files`	Lists the files in the specified directory.

Command	Description
`show host`	Displays host information for hosts reachable by the router.
`show interfaces`	Displays information about interfaces.
`show ntp associations`	Shows the status of configured NTP servers.
`show system boot-messages`	Displays boot messages generated by the kernel.
`show system connections`	Displays active network connections on the system.
`show system kernel-messages`	Displays messages in the kernel ring buffer.
`show system memory`	Displays system memory usage.
`show system processes`	Displays active system processes.
`show system storage`	Displays system file system usage and available storage space.
`show version`	Displays information about the version of router software.

This section includes the following examples:

- Example 1-13 Showing the system host name
- Example 1-14 Showing the system DNS IP
- Example 1-15 Showing the system date and time
- Example 1-16 Showing NTP server configuration and connection status

NOTE *The output in these examples may show information unrelated to the sample configurations.*

Showing Host Information

To view the configured host name, use the **show host name** command in operational mode, as shown in Example 1-13:

Example 1-13 Showing the system host name

```
vyatta@R1> show host name
R1
vyatta@venus>
```

To Show host by name, and see which IP address has been used for DNS resolution, use the **show host** *hostname* command in operational mode, as shown in Example 1-14:

Example 1-14 Showing the system DNS IP

```
vyatta@R1> show host R1
Server: 172.16.0.34
Address: 172.16.0.34#53

Name:   R1.mydomain.com
Address: 10.0.0.65

vyatta@R1>
```

Showing the Date and Time

To view the time according to the system clock, use the **show host date** command in operational mode, as shown in Example 1-15:

Example 1-15 Showing the system date and time

```
vyatta@R1> show host date
Tue Apr 24 22:23:07 PDT 2007
vyatta@R1>
```

Showing NTP Server Configuration

To see the NTP servers configured for the router and see their connection status, use the **show ntp associations** command in operational mode. To reduce the amount of time needed to resolve DNS names, use the **no-resolve** option, as shown in Example 1-16:

Example 1-16 Showing NTP server configuration and connection status

```
vyatta@R1> show ntp associations no-resolve
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
176.16.0.42   0.0.0.0     16 u    - 1024    0    0.000    0.000 4000.00
10.10.40.42   0.0.0.0     16 u    - 1024    0    0.000    0.000 4000.00
vyatta@R1>
```

Chapter 2: Ethernet and VLAN Interfaces

This chapter explains how to configure Ethernet interfaces, the loopback interface, and VLAN interfaces.

The following topics are covered:

- Ethernet Interfaces Overview
- Configuring Physical Ethernet Interfaces
- Configuring the Loopback Interface
- Configuring VLANs
- Monitoring Ethernet Interfaces

Ethernet Interfaces Overview

A router receives packets via its network interfaces from its neighboring routers. Some of those packets will be destined for the router itself, but most of them will normally be forwarded on via another network interface to another router or to locally connected hosts.

There are many different types of interfaces. This section deals with defining configuration for Ethernet interfaces.

Virtual Interfaces (Vifs)

VLANs are identified by a 4-byte tag that is inserted in the front of the Layer 2 Ethernet header. Having this additional tag means that interfaces configured for 802.1q are not compatible with standard Ethernet packets.

Like a physical Ethernet interface, each vif can have multiple addresses assigned to it. If you are using 802.1q VLANs, create vif configuration nodes beneath the physical interface and assign the IP address to the vif. If you are not using 802.1q, but you want to have multiple networks on the same physical interface (that is, you want to use multinetting, but not VLANs), simply create multiple **address** configuration nodes directly under the physical interface, without using vifs.

In the Vyatta system, an Ethernet interface may be used simultaneously as a standard port and an 802.1q port. To do this, configure a vif for the interface, and assign the VLAN ID for the interface to the vif. On Ethernet interfaces, a vif is always a VLAN interface, and its identifier is the VLAN ID.

This feature may not be compatible with all Ethernet switches: some switches require a physical Ethernet interface to be exclusively either a 802.1q interface or a standard Ethernet interface.

Enabling Interfaces

The Vyatta system can only use interfaces that are available to the operating system kernel (that is, interfaces that physically exist on the system) and have been created in the configuration tree.

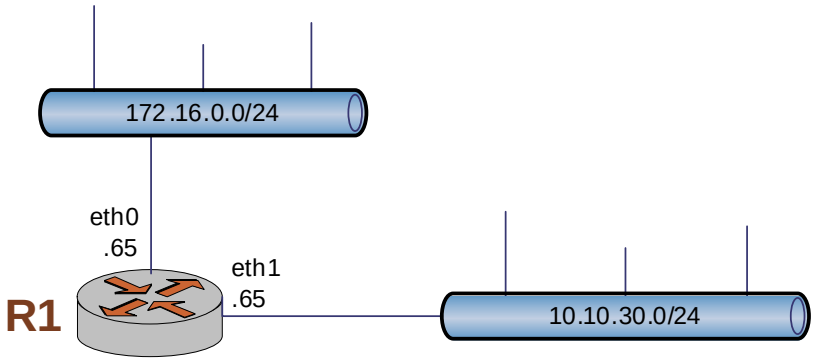
The Vyatta system automatically creates configuration nodes for all available physical interfaces on startup.

If you want to use an interface with a specific function (say, BGP) the interface must be enabled within the configuration node for that function (for example, within the BGP configuration node.

Configuring Physical Ethernet Interfaces

In this section, sample configurations are presented for Ethernet interfaces. The sample configuration is shown in Figure 2-1.

Figure 2-1 Ethernet interfaces



This section includes the following examples:

- Example 2-1 Viewing available system interfaces
- Example 2-2 Creating and configuring Ethernet interfaces

Viewing Available Interfaces

You can only configure interfaces that actually are available to the operating system on the hardware you are using. To view all the interfaces known to the operating system, use the **show interfaces system** command in operational mode, as shown in Example 2-1:

Example 2-1 Viewing available system interfaces

```
vyatta@vyatta> show interfaces system
```

Configuring Ethernet Interfaces

In the Vyatta system router, most configuration can be applied either directly to the physical interface, or to a *virtual interface* (vif), which is a logical interface created for the physical interface. When the router starts up, it automatically detects the physical interfaces available on your device and creates configuration nodes for them. For example, on a system with two Ethernet interfaces, the router automatically creates configuration nodes for **eth0** and **eth1**.

Ethernet vifs are used only when 802.1Q VLANs are to be supported. In a basic Ethernet configuration, such as that for trial or evaluation or for a simple network topology, it will often be simplest and adequate to apply IP addresses directly to the physical interface.

Each physical interface can have multiple IP addresses assigned to it.If you want to have multiple networks on the same physical interface (that is, if you want to use multinetting, but not VLANs), simply create multiple **address** configuration nodes directly under the primary interface.

This sequence applies IP addresses directly to the two Ethernet interfaces already configured for the system—eth0 and eth1. These interfaces were automatically created by the system on startup, when the system detected the physical interfaces. Each IP address is applied directly to the interface.

To create and configure Ethernet interfaces, perform the following steps in configuration mode:

Example 2-2 Creating and configuring Ethernet interfaces

Step	Command
Create the configuration node for eth0. You can create the node completely down to the first IP address, and set the prefix length, with the same **set** statement.	`vyatta@R2# set interfaces ethernet eth0 address 172.16.0.65 prefix-length 24` `[edit]`
Assign an IP address to interface eth1.	`vyatta@R1# set interfaces ethernet eth1 address 10.10.30.65 prefix-length 24` `[edit]`

Example 2-2 Creating and configuring Ethernet interfaces

Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show interfaces`
	`  loopback lo {`
	`  }`
	`  ethernet eth0 {`
	`  address 172.16.0.65 {`
	`  prefix-length: 24`
	`  }`
	`  }`
	`  ethernet eth1 {`
	`  address 10.10.30.65 {`
	`  prefix-length: 24`
	`  }`
	`  }`
	
	`[edit]`
	`vyatta@R1#`

Configuring the Loopback Interface

The loopback interface is a special software-only interface that emulates a physical interface and allows the router to “connect” to itself. Packets routed to the loopback interface are rerouted back to the router and processed locally. Packets routed out the loopback interface but not destined for the loopback interface are dropped.

The loopback interface provides a number of advantages:

- As long as the router is functioning, the loopback interface is always up, and so is very reliable. As long as there is even one functioning link to the router, the loopback interface can be accessed. The loopback interface thus eliminates the need to try each IP address of the router until you find one that is still up.
- Because the loopback interface is always up, a routing session (such as a BGP session) can continue even if the outbound interface fails.
- You can simplify collection of management information by specifying the loopback interface as the interface for sending and receiving management information such as logs and SNMP traps.
- The loopback interface can be used as to increase security, by filtering incoming traffic using access control rules that specify the local interface as the only acceptable destination.

- In OSPF, you can advertise a loopback interface as an interface route into the network, regardless of whether physical links are up or down. This increases reliability, since the the routing traffic is more likely to be received and subsequently forwarded.
- In BGP, parallel paths can be configured to the loopback interface on a peer device. This provides improved load sharing.

The router automatically creates the loopback interface on startup, with an interface name of **lo**. You must configure an IP address for the interface. The IP address for the loopback interface must be unique, and must not be used by any other interface.

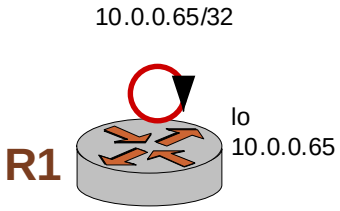
When configuring the router, it is good practice to take advantage of the loopback interface's reliability:

- The router's hostname should be mapped to the loopback interface address, rather than a physical interface.
- In OSPF and BGP, the router ID should be set to the loopback address.
- The network for the loopback interface can be small, since IP address space is not a consideration in this cse. Often a prefix of /32 is assigned.

NOTE *In some systems, the IP address 127.0.0.0 is assigned to the loopback interface by convention. However, in the Vyatta system the network 127.0.0.0/8 is reserved for XORP to communicate between processes. As a result, no IP address on this reserved network may be configured on any interface. Any other network may be assigned to the loopback interface.*

This sequence assigns the address 10.0.0.65 to the loopback interface. The subnet is set to /32. When you have finished, the interface will be configured as in Figure 2-3.

Figure 2-2 Configuring the loopback interface



To create and configure the loopback interface, perform the following steps in configuration mode:

Example 2-3 Configuring the loopback interface

Step	Command
Assign the IP address to the loopback interface	vyatta@R1# **set interfaces loopback lo address 10.0.0.65 prefix-length 32** [edit]

Commit and view the configuration.	vyatta@R1# **commit**
	OK
	[edit]
	vyatta@R1# **show interfaces**
	loopback lo {
	address 10.0.0.65 {
	prefix-length: 32
	}
	}
	ethernet eth0 {
	address 172.16.0.65 {
	prefix-length: 24
	}
	}
	ethernet eth1 {
	address 10.10.30.65 {
	prefix-length: 24
	}
	}
	
	[edit]
	vyatta@R1#

Configuring VLANs

VLANs are identified by a 4-byte tag that is inserted in the front of the Layer 2 Ethernet header. Having this additional tag means that interfaces configured for 802.1q are not compatible with standard Ethernet packets.

Like a physical Ethernet interface, each vif can have multiple addresses assigned to it. If you are using 802.1q VLANs, create vif configuration nodes beneath the physical interface and assign the IP address to the vif. If you are not using 802.1q, but you want to have multiple networks on the same physical interface (that is, you want to use multinetting, but not VLANs), simply create multiple **address** configuration nodes directly under the physical interface, without using vifs.

In the Vyatta system, an Ethernet interface may be used simultaneously as a standard port and an 802.1q port. To do this, configure a vif for the interface, and assign the VLAN ID for the interface to the vif. On Ethernet interfaces, a vif is always a VLAN interface, and its identifier is the VLAN ID.

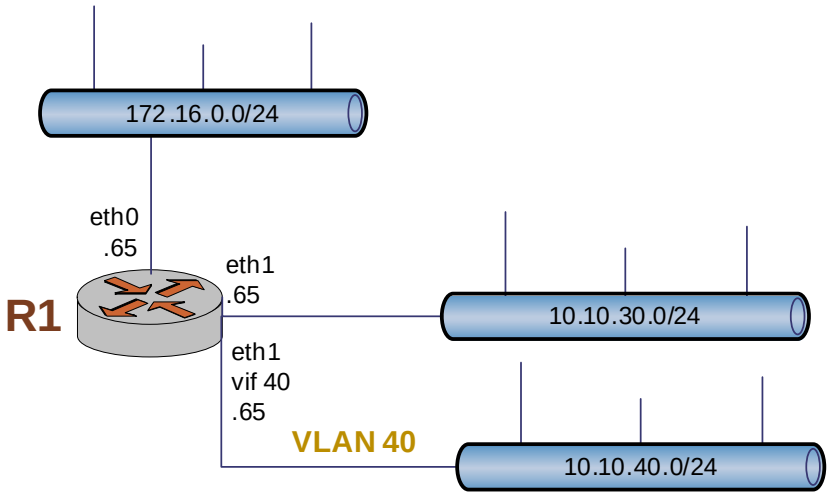
This feature may not be compatible with all Ethernet switches: some switches require a physical Ethernet interface to be exclusively either a 802.1q interface or a standard Ethernet interface.

This sequence configures a VLAN interface on router R1—vif 40 on eth1. The vif identifier is the VLAN ID, and this vif connects to VLAN 40. After configuring this VLAN, router R1 will have:

- One interface (eth0) that is configured as only a standard Ethernet interface
- One interface (eth1) that is configured as both a standard interface (IP address 10.10.30.65 applied directly to the interface) and as an 802.1q interface with one logical VLAN interface (IP address 10.10.40.65 applied to vif 40).

When you have finished, the interfaces will be configured as in Figure 2-3.

Figure 2-3 VLAN configuration



To create and configure a VLAN interface, perform the following steps in configuration mode:

Example 2-4 Creating and configuring Ethernet interfaces

Step	Command
Create the configuration node for vif 40 on eth1 and assign an IP address.	`vyatta@R1# set interfaces ethernet eth1 vif 40 address 10.10.40.65 prefix-length 24` `[edit]`

Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]vyatta@R1# show interfaces`
	`loopback lo {`
	`address 10.0.0.65 {`
	`prefix-length: 32`
	`}`
	`}`
	`ethernet eth0 {`
	`address 172.16.0.65 {`
	`prefix-length: 24`
	`}`
	`}`
	`ethernet eth1 {`
	`address 10.10.30.65 {`
	`prefix-length: 24`
	`}`
	`vif 40 {`
	`address 10.10.40.65 {`
	`prefix-length: 24`
	`}`
	`}`
	`}`
	
	`[edit]`
	`vyatta@R1#`

When you refer to a vif within an **interfaces ethernet** command (such as **set interfaces ethernet** or **show interfaces ethernet**) you refer to it as **ethernet** *int-name* **vif** *vif-id*, as shown in the following example:

```
show interfaces ethernet eth1 vif 40
```

When you refer to the same vif within other commands, you refer to it as *int.vif*, as shown in bold in the following example:

```
set protocols rip interface eth1.40 address 10.10.40.65
```

Monitoring Ethernet Interfaces

You can use the following operational commands to monitor Ethernet interfaces. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`show interfaces ethernet`	Shows information for Ethernet interfaces.
`show interfaces system`	Shows information for all interfaces available to the Linux kernel.

Chapter 3: Serial Interfaces

This chapter describes how to configure serial interfaces on the Vyatta system.

The following topics are covered:

- Serial Interfaces Overview
- Configuring Serial Interfaces
- Monitoring Serial Interfaces

Serial Interfaces Overview

A router receives packets via its network interfaces from its neighboring routers. Some of those packets will be destined for the router itself, but most of them will normally be forwarded on via another network interface to another router or to locally connected hosts.

There are many different types of interface. This section deals with defining configuration for serial interfaces.

Virtual Interfaces (Vifs)

The Vyatta system router distinguishes between physical interfaces (*interfaces*), and logical interfaces (*virtual interfaces*, or *vifs*).

Every physical network device in the system is considered to be an "interface." An example of a interface is a physical port on a serial card. Every serial interface has zero or more corresponding vifs.

On serial interfaces, physical line characteristics are specific for the interface, but encapsulation (Cisco HDLC, Frame Relay, or Point-to-Point Protocol) is specified for vifs.

Unlike Ethernet interfaces, a physical serial interface cannot directly have a configured IP address. Instead, the IP address must be assigned to the vif.

Note that each serial vif can support exactly one IP address

Enabling Interfaces

The Vyatta system can only use interfaces that are available to the operating system kernel (that is, interfaces that physically exist on the system) and have been created in the configuration tree and configured with an IP address.

The Vyatta system automatically creates configuration nodes for all available physical interfaces on startup.

If you want to use an interface with a specific function (say, BGP) the interface must be enabled within the configuration node for that function (for example, within the BGP configuration node).

Configuring Serial Interfaces

The system will automatically discover any available physical serial interfaces on startup. Before you can apply any configuration to a serial interface, a vif must be “created” for the interface and its encapsulation specified in the configuration tree.

For serial interfaces, physical line characteristics are applied to the interface as a whole. Encapsulation characteristics are applied to the vif, as shown in the configuration hierarchy below:

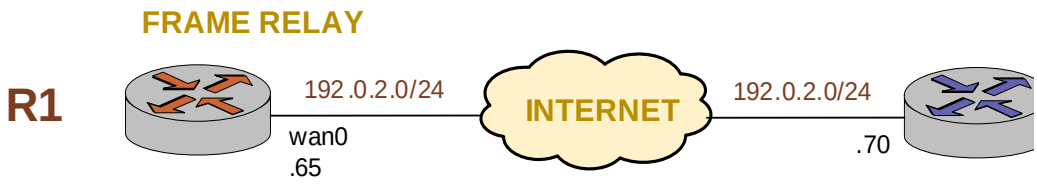
```
interfaces {
   serial wan0 {
      ppp {
         vif 1 {
         }
      }
   }
}
```

The current implementation supports Cisco HDLC, Frame Relay, and Point-to-Point Protocol encapsulation.

- Cisco HDLC and point-to-point interfaces support only one vif, and this vif must have the identifier “1”.
- The identifier for Frame Relay vifs is the DLCI number. This can range from 16 through 991.
- Currently, any vif on a serial interface can support exactly one IP address.

In this section, a sample Frame Relay configuration is presented for a serial interface on router R1. The sample configuration is shown in Figure 3-1.

Figure 3-1 Network interfaces



This section includes the following examples:

- Example 3-1 Viewing available system interfaces
- Example 3-2 Creating and configuring a Frame Relay interface

Viewing Available Interfaces

You can only configure interfaces that actually are available to the operating system on the hardware you are using.

To view all the interfaces known to the operating system, use the **show interfaces system** command in operational mode, as shown in Example 3-1:

Example 3-1 Viewing available system interfaces

```
vyatta@R1> show interfaces system
```

Configuring a Frame Relay Interface

Example 3-2 sets up a Frame Relay interface on interface wan0. In this example:

- A Sangoma A101 T1/E1 serial card is connected to the interface.
- The physical line is a T1 line.
- The interface has one vif, which has DLCI number 16.
- The local IP is 192.0.2.65. This is in the public IP range, since this interface will connect over the wide-area network.
- The IP address of the far end is 192.0.2.70. This is on the same network (prefix-length 24) as this interface.

Tip: *Where public IP addresses would normally be used, the example uses RFC 3330 "TEST-NET" IP addresses (192.0.2.0/24)*

To create and configure this serial interface, perform the following steps in configuration mode:

Example 3-2 Creating and configuring a Frame Relay interface

Step	Command
Specify the kind of physical line the interface will be using.	`vyatta@R1# set interfaces serial wan0 t1-options` `[edit]`
Set the line encapsulation to Frame Relay.	`vyatta@R1# set interfaces serial wan0 encapsulation frame-relay` `[edit]`

Example 3-2 Creating and configuring a Frame Relay interface

Create a vif with DCLI 16.	`vyatta@R1# set interfaces serial wan0 frame-relay vif 16` `[edit]`
Assign the local IP address to the vif.	`vyatta@R1# set interfaces serial wan0 frame-relay vif 16 address local-address 192.0.2.65` `[edit]`
Set the network mask (prefix length) for the vif.	`vyatta@R1# set interfaces serial wan0 frame-relay vif 16 address prefix-length 24` `[edit]`
Set the IP address of the far end of the connection.	`vyatta@R1# set interfaces serial wan0 frame-relay vif 16 address remote-address 192.0.2.70` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

View the configuration.	`vyatta@R1# exit`
	`[edit]`
	`vyatta@R1> show interfaces serial wan0`
	`encapsulation: "frame-relay"`
	`description: "sangoma-t1/e1"`
	`t1-options {`
	`}`
	`frame-relay {`
	`vif 16 {`
	`address {`
	`local-address: 192.0.2.65`
	`prefix-length: 24`
	`remote-address: 192.0.2.70`
	`}`
	`}`
	`}`
	`vyatta@R1>`

Monitoring Serial Interfaces

You can use the following operational commands to monitor serial interfaces. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`show interfaces serial`	Shows information for serial interfaces.
`show interfaces system`	Shows information for interfaces available to the Linux kernel.

Chapter 4: Basic Services

This chapter explains how to configure system services such as DHCP, HTTP, Telnet, and SSH.

The following topics are presented:

- Basic Services Overview
- Configuring DHCP
- Configuring HTTP, Telnet, and SSH
- Monitoring Service Information

Basic Services Overview

This section presents the following topics:

- DHCP
- Access Protocols: HTTP, SSH, and Telnet

DHCP

Dynamic Host Configuration Protocol (DHCP) allows dynamic assignment of reusable IP addresses and other configuration information to DHCP clients. This reduces costs, configuration effort, and management burden associated with Internet access. On the other hand, it also increases network and service overhead.

In DHCP, the server assigns an IP address and other configuration parameters to a client for a limited period of time. This period of time is called the *lease*. The lease is valid for the period you configure on the router, or until the client explicitly relinquishes the address.

To use the DHCP service, you define a pool of IP addresses for each subnet assigned by the DHCP server. Each DHCP address pool is associated with an Ethernet interface on the router. For each address pool, you can specify the length of time an address will be valid (its lease duration). The default lease duration is 24 hours. You can also specify the DNS and WINS servers available to clients on the subnet.

You can exclude addresses from an address pool, to reserve IP addresses for specific network devices. You can also statically map an IP address to the MAC address of a device. The DHCP service listens on UDP port 67 for lease requests from DHCP clients. The request packet allows the router to determine which interface the client is located on. It then assigns an IP from the appropriate pool and binds it to the client.

The router supports DHCP relay.

A DHCP relay agent receives DHCP packets from DHCP clients and forwards them to a DHCP server. This allows you to place DHCP Clients and DHCP servers on different networks; that is, across router interfaces.

The relay agent is configured with addresses of DHCP servers to which they should relay client DHCP message. The relay agent intercepts the broadcast, sets the gateway address (the **giaddr** field of the DHCP packet) and, if configured, inserts the Relay Agent Information option (option 82) in the packet and forwards it to the DHCP server.

The DHCP server echoes the option back verbatim to the relay agent in server-to-client replies, and the relay agent strips the option before forwarding the reply to the client.

Access Protocols: HTTP, SSH, and Telnet

The Vyatta system router supports access from remote systems by means of HTTP, Telnet, and SSH.

- HTTP provides remote web access to the Vyatta system.
- The Secure Shell (SSH) protocol provides secure encrypted communications between two hosts communicating over an insecure network. The Vyatta system supports both the SSH v1 and v2 protocols.
  - For SSH v1, the software supports encrypted communication using ciphers such as 3DES or Blowfish, as selected by the SSH client. The default is 3DES.
  - For SSH v2, the software supports 128-bit AES, Blowfish, 3DES, CAST128, Arcfour, 192-bit AES, or 256-bit AES, as selected by the SSH client.
- Telnet provides unencrypted communications between the router and another host. If you use SSH, we recommend that you disable Telnet access, which is not secure.

You can leave the port as the default, which is the well-known port for the protocol, or configure a non-standard port.

For security reasons, remote access to the router is disabled by default. You must configure the router explicitly (by creating the SSH and Telnet service configuration nodes) so that users on remote systems can access it.

Configuring DHCP

This section includes the following examples:

- Example 4-1 Enabling the DHCP service
- Example 4-2 Configuring DHCP address pools
- Example 4-3 Setting up DHCP relay

Enabling DHCP

To use DHCP on the Vyatta system, you must enable the DHCP service.

To enable the DHCP service, perform the following steps in configuration mode:

Example 4-1 Enabling the DHCP service

Step	Command
Enable DHCP.	`vyatta@R1# set service dhcp` `[edit]`

Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show service`
	`  dhcp {`
	`  }`
	
	`[edit]`

Configuring DHCP Address Pools

Configure DHCP address pools if you want the router to act as a DHCP server for the network.

Example 4-2 creates three address pools:

- **ETH0_POOL**. This address pool serves subnet 172.16.0.0/24, which is connected to the router interface eth0. The lease time will remain at the default, 24 hours (86,400 seconds). This address pool is able to use the DNS name server at 172.16.0.34.
- **ETH1_30_POOL**. This address pool serves subnet 10.10.30.0/24, which is connected directly to interface eth1. The lease time will remain at the default, 24 hours (86,400 seconds). This address pool will use the DNS name server at 10.10.40.34, which is directly connected to eth1.40 (that is, eth1 vif 40).
- **ETH1_40_POOL**. This address pool serves subnet 10.10.40.0/24, which is also connected to interface eth1.40. The lease time will remain at the default, 24 hours (86,400 seconds). This address pool will use the DNS name server at 10.10.40.34, which is connected to eth1.40.

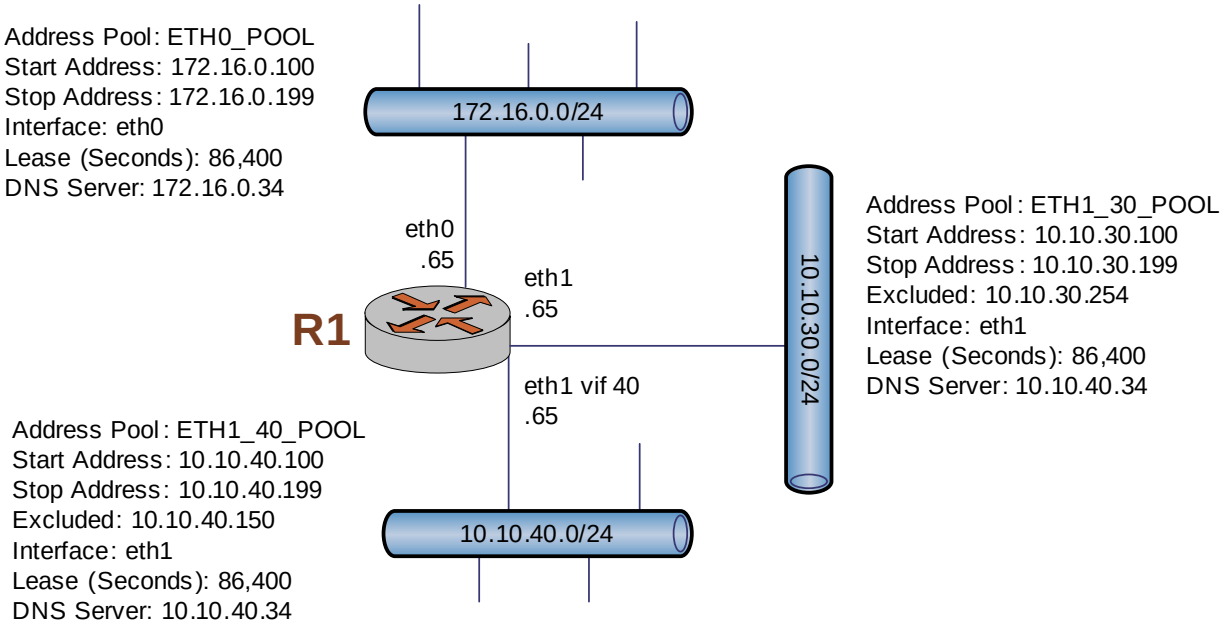
In all of these pools, the range of addresses is configured for .100 through .199.

- The addresses for the network routers in the sample topology, and for the servers configured in “Chapter 1: System Management,” fall below this range.

- The address for the default gateway (.254) and the broadcast address (.255) fall above it.
- As an example of specific address exclusion, the address pool ETH1_40_POOL excludes one additional address: 10.10.40.150.

Figure 4-1 shows the sample address pool configuration.

Figure 4-1 DHCP



To configure DHCP address pools, perform the following steps in configuration mode:

Example 4-2 Configuring DHCP address pools

Step	Command
Create the configuration node for ETH0_POOL. Specify the start and stop IP addresses for the pool.	`vyatta@R1# set service dhcp-server name ETH0_POOL start 172.16.0.100 stop 172.16.0.199` `[edit]`
Specify the size of the network served by ETH0_POOL.	`vyatta@R1# set service dhcp-server name ETH0_POOL network-mask 24` `[edit]`

Example 4-2 Configuring DHCP address pools

Bind ETH0_POOL to interface eth0.	vyatta@R1# **set service dhcp-server name ETH0_POOL interface eth0** [edit]
Specify a DNS server for ETH0_POOL.	vyatta@R1# **set service dhcp-server name ETH0_POOL dns-server 172.16.0.34** [edit]
Create the configuration node for ETH1_30_POOL. Specify the start and stop IP addresses for the pool.	vyatta@R1# **set service dhcp-server name ETH1_30_POOL start 10.10.30.100 stop 10.10.30.199** [edit]
Specify the size of the network served by ETH1_30_POOL.	vyatta@R1# **set service dhcp-server name ETH1_30_POOL network-mask 24** [edit]
Bind ETH1_30_POOL to interface eth1.	vyatta@R1# **set service dhcp-server name ETH1_30_POOL interface eth1** [edit]
Specify a DNS server for ETH1_30_POOL.	vyatta@R1# **set service dhcp-server name ETH1_30_POOL dns-server 10.10.40.34** [edit]
Create the configuration node for ETH1_40_POOL. Specify the start and stop IP addresses for the pool.	vyatta@R1# **set service dhcp-server name ETH1_40_POOL start 10.10.40.100 stop 10.10.40.199** [edit]
Specify the size of the network served by ETH1_40_POOL.	vyatta@R1# **set service dhcp-server name ETH1_40_POOL network-mask 24** [edit]
Exclude address 10.10.40.150 from the address pool.	vyatta@R1# **set service dhcp-server name ETH1_40_POOL exclude 10.10.40.150** [edit]
Bind ETH1_40_POOL to interface eth1.	vyatta@R1# **set service dhcp-server name ETH1_40_POOL interface eth1** [edit]
Specify a DNS server for ETH1_40_POOL.	vyatta@R1# **set service dhcp-server name ETH1_40_POOL dns-server 10.10.40.34** [edit]

Example 4-2 Configuring DHCP address pools

Commit and view the configuration.

```
vyatta@R1# commit
OK
[edit]
vyatta@R1# show service dhcp-server
   name "ETH0_POOL" {
      start 172.16.0.100 {
         stop: 172.16.0.199
      }
      network-mask: 24
      dns-server 172.16.0.34
      interface: "eth0"
   }
   name "ETH1_30_POOL" {
      start 10.10.30.100 {
         stop: 10.10.30.199
      }
      network-mask: 24
      dns-server 10.10.40.34
      interface: "eth1"
   }
   name "ETH1_40_POOL" {
      start 10.10.40.100 {
         stop: 10.10.40.199
      }
      exclude 10.10.40.150
      network-mask: 24
      dns-server 10.10.40.34
      interface: "eth1"
   }

[edit]
```

Setting Up DHCP Relay

Configure DHCP relay if you want the router to forward DHCP relay to another DHCP server.

Example 4-3 does the following:

- Directs the router to forward client-to-server DHCP messages out through interface eth0 to the DHCP server at 172.16.1.52.
- Enables relay options. This directs the router to add the Relay Agent Information option (option 82) to the DHCP message before forwarding, as specified by RFC 3046.
- Re-forwarding of DHCP messages will not be permitted by this router. If a packet is received that already contains relay information, the packet is discarded.
- Other relay option parameters are left at default values. This means that the router will use port 67 for DHCP messaging, will allow a maximum DHCP packet size of at most 576 bytes, and will have a maximum hop count of 10 hops.

To configure DHCP relay, perform the following steps in configuration mode:

Example 4-3 Setting up DHCP relay

Step	Command
Enable DHCP relay on interface eth0, and specify the IP address of the DHCP server.	`vyatta@R1# set service dhcp relay interface eth0 server 172.16.1.52` `[edit]`
Enable relay options.	`vyatta@R1# set service dhcp relay interface eth0 relay-options` `[edit]`
Set the router to discard messages containing relay information. Leave other parameters at default values.	`vyatta@R1# set service dhcp relay interface eth0 relay-options relay-agents-packets discard` `[edit]`

Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show service dhcp relay`
	`  interface eth0 {`
	`  server 172.16.1.52`
	`  relay-options {`
	`  relay-agents-packets: "discard"`
	`  }`
	`  }`
	
	`[edit]`

Configuring HTTP, Telnet, and SSH

In this section, sample configurations are presented for HTTP, Telnet, and SSH. This section includes the following examples:

- Example 4-4 Enabling HTTP access
- Example 4-5 Enabling SSH access
- Example 4-6 Enabling Telnet access

HTTP

Configuring HTTP is optional, but creating the HTTP service will provide remote access to the router.

By default, HTTP is enabled on port 80. Example 4-4 enables HTTP on port 8080, as shown in Figure 4-2.

Figure 4-2 Enabling HTTP access

To enable the HTTP service on the router, perform the following steps in configuration mode:

Example 4-4 Enabling HTTP access

Step	Command
Create the configuration node for the HTTP service and specify the port number.	`vyatta@R1# set service http port 8080` `[edit]`

Commit the information.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Example 4-4 Enabling HTTP access

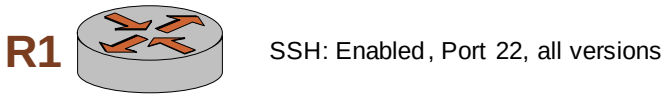
Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show service http`
	`  port: 8080`
	
	`[edit]`

SSH

Configuring SSH is optional, but creating the SSH service will provide secure remote access to the router.

Example 4-5 enables SSH on the default port (port 22), as shown in Figure 4-4. By default, only SSH version 2 is enabled, but Example 4-5 enables SSH for all versions of SSH.

Figure 4-3 Enabling SSH access



To enable the SSH service on the router, perform the following steps in configuration mode:

Example 4-5 Enabling SSH access

Step	Command
Create the configuration node for the SSH service.	`vyatta@R1# set service ssh protocol-version all` `[edit]`

Commit and view the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show service ssh`
	`  protocol-version: "all"`
	
	`[edit]`

Telnet

Configuring Telnet is optional, but creating the Telnet service will allow you to access the router remotely. Example 4-6 enables Telnet on the default port (port 23), as shown in Figure 4-4.

Figure 4-4 Enabling Telnet access

To enable the Telnet service on the router, perform the following steps in configuration mode:

Example 4-6 Enabling Telnet access

Step	Command
Create the configuration node for the Telnet service.	`vyatta@R1# set service telnet` `[edit]`

Commit the information.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Monitoring Service Information

This section presents the following topic:

- Service Operational Commands

Service Operational Commands

You can use the following operational commands to monitor service information. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`clear dhcp leases`	Removes current DHCP leases.
`show dhcp leases`	Displays current DHCP lease information.
`show dhcp statistics`	Displays DHCP server statistics.

This section includes the following examples:

- Example 4-7 Viewing DHCP lease information
- Example 4-8 Viewing DHCP statistics

Viewing DHCP Lease Information

To view DHCP lease information, use the **show dhcp lease** command in operational mode, as shown in Example 4-7. Example 4-7 uses the **pool** option, and will display lease information for address pool ETH1_40_POOL.

Example 4-7 Viewing DHCP lease information

```
vyatta@R1> show dhcp leases pool ETH1_40_POOL
```

Viewing DHCP Statistics

To view DHCP lease information, use the **show dhcp statistics** command in operational mode, as shown in Example 4-7:

Example 4-8 Viewing DHCP statistics

```
vyatta@R1> show dhcp statistics ETH1_40_POOL

req ct    resp ct   tot addr pool   num leased addr    num avail addr
------    -------   -------------   ---------------    --------------
0         0         0               0                  0
```

Chapter 5: Forwarding and Routing

This chapter provides a brief overview of basic routing topics: traffic fowarding, unicast routing, multicast routing, and multicast topology discovery.

The following topics are covered:

- Forwarding
- Unicast Routing Overview
- Multicast Routing Overview
- Multicast Topology Discovery
- Route Selection Process
- Configuring Forwarding
- Displaying Route Information

Forwarding

The forwarding engine of a router is that part that receives packets from one interface and forwards them through another interface. On the Vyatta system, the configuration interface for the forwarding engine is called the Forwarding Engine Abstraction (**fea**) for unicast packets, and the Multicast Forwarding Engine Abstraction (**mfea**) for multicast packets.

You might want a router in one situation to have a different forwarding policy from that in another. For example, you might want the router to forward unicast packets but not multicast packets. Configuring the forwarding engine allows you to selectively enable or disable forwarding for unicast and multicast.

On the Vyatta system:

- Unicast packet forwarding is enabled by default for each protocol when the protocol configuration node is created.
- Multicast packet forwarding must be explicitly enabled.

Unicast Routing Overview

To forward packets, a router maintains a forwarding table containing routes indicating to which neighboring router a packet for a particular destination should be forwarded. At the minimum, then, a route then consists of a destination subnet and a next hop.

The destination subnet is usually represented as a base IP address and a prefix-length in bits. For example, the subnet 128.16.64.0/24 has a prefix length of 24 bits, indicating that the first 24 bits of this address identify the network in question, and the last 8 bits identify hosts on this subnet. Thus, a route for this subnet would be used to forward packets for addresses 128.16.64.0 to 128.16.64.255 inclusive. The next hop can be the IP address of a neighboring router, or it might indicate that the route is for a subnet that is directly connected to this router.

Prefix Matching

IP routers perform longest prefix match forwarding. This means that a router might have more than one route that matches a destination address, and under such circumstances, it will use the route that has the longest prefix. For example, suppose a router has the following two routes:

- Subnet: 128.16.0.0/16, next hop: 10.0.0.1
- Subnet: 128.16.64.0/24, next hop: 10.0.0.2

A packet destined for 128.16.0.1 would match the first route only, and so would be forwarded to 10.0.0.1. However, a packet destined for 128.16.64.1 would match both routes, and so would be forwarded to 10.0.0.2 because the second route has a longer prefix (24 is longer than 16).

Populating the Forwarding Table

To be useful, a router needs to populate its forwarding table. It does this in three ways:

- Routes for directly connected subnets are automatically entered into the forwarding table.
- Routes may be configured via the router's configuration file or command line interface. Such routes are known as *static routes*.
- Routes may be learned from another router via a routing protocol. Such routes are known as *dynamic routes*.

Static Routes

Static routes are discussed in detail in "Chapter 7: Static Routes."

Dynamic Routing

Many different routing protocols can supply dynamic routes. The following unicast dynamic routing protocols are supported by the Vyatta system:

- **Routing Information Protocol (RIP).** This is probably the simplest intra-domain routing protocol, and is often used on small networks. RIP is discussed in detail in "Chapter 8: RIP"
- **Open Shortest Path First (OSPF).** Used for intra-domain routing, often on large ISP networks. OSPF is discussed in more detail in "Chapter 9: OSPF"
- **Border Gateway Protocol (BGP).** BGP is used for inter-domain routing. BGP is discussed in detail in "Chapter 10: BGP"

In addition, there are also multicast routing protocols. Of these, the Vyatta system supports PIM-SM and IGMP.

Route Redistribution

A common requirement is to redistribute routes between routing protocols. Examples of situations where routes might be redistributed are as follows:

- When interconnecting some statically routed subnets with some subnets using IP for dynamic routing.

Rather than configure the static routes, and additionally informing RIP to originate route advertisements for the same subnets, it is simpler and less error-prone to configure the router to redistribute all the static routes into RIP.

- When a network uses RIP internally, but also uses BGP to peer with the rest of the Internet.

  One solution would be to configure BGP at the border routes to originate route advertisements for the internal subnets. However, if a new subnet is added internally, then the border routers also need to be correctly modified. Instead, you can simply configure the border routers to redistribute RIP routes into BGP.

The Vyatta system redistributes routes using routing policies. Routing policies are discussed in detail in "Chapter 6: Routing Policies."

Multicast Routing Overview

IP multicast is a technology that allows one-to-many and many-to-many distribution of data on the Internet. Senders send their data to a multicast IP destination address, and receivers express an interest in receiving traffic destined for such an address. The network then determines how to get the data from senders to receivers.

If both the sender and receiver for a multicast group are on the same local broadcast subnet, the routers do not need to intervene in the process, and communication can take place directly. If, however, the sender and receiver are on different subnets, then a multicast routing protocol needs to be involved in setting up multicast forwarding state on the tree between the sender and the receivers.

Multicast Routing Protocols

Broadly speaking, there are two different types of multicast routing protocols:

- Dense-mode protocols

  In these, traffic from a new multicast source is delivered to all possible receivers. Then, subnets where there are no members request to be pruned from the distribution tree.

- Sparse-mode protocols

  In these, explicit control messages are used to ensure that traffic is only delivered to the subnets where there are receivers that have requested to receive it.

Examples of dense-mode protocols are DVMRP and PIM Dense Mode. Examples of sparse-mode protocols are PIM Sparse Mode, CBT, and MOSPF. Most of these protocols are largely historic at this time, with the exception of PIM Sparse Mode (PIM-SM) and PIM Dense Mode (PIM-DM). At this time, PIM-DM is not very widely used.

In addition to the routing protocols used to set up forwarding state between subnets, a way is needed for the routers to discover that there are local receivers on a directly attached subnet. For IPv4 this role is served by the Internet Group Management Protocol (IGMP).

Service Models: ASM vs. SSM

There are two different models for IP multicast:

- Any Source Multicast (ASM), in which a receiver joins a multicast group, and receives traffic from any senders that send to that group.
- Source-Specific Multicast (SSM), in which a receiver explicitly joins to a (source, group) pairing.

Traditionally IP multicast used the ASM model, but problems deploying inter-domain IP multicast resulted in the much simpler SSM model being proposed. In the future it is likely that ASM will continue to be used within intranets and enterprises, but SSM will be used when multicast is used inter-domain. The two models are compatible, and PIM-SM can be used as a multicast routing protocol for both. The principal difference is that ASM only requires IGMPv2 or MLDv1, whereas SSM requires IGMPv3 or MLDv2 to permit the receivers to specify the address of the sending host.

Multicast Addresses

For IPv4, multicast addresses are in the range 224.0.0.0 to 239.255.255.255 inclusive. Addresses within 224.0.0.0/24 are considered link-local and should not be forwarded between subnets. Addresses within 232.0.0.0/8 are reserved for SSM usage. Addresses in 239.0.0.0/8 are ASM addresses defined for varying sizes of limited scope.

Supported Protocols

Vyatta supports the following multicast protocols:

- PIM Sparse Mode for both ASM and SSM multicast routing for IPv4.
- IGMPv1 and IGMPv2 for IPv4 local multicast membership.

Multicast Topology Discovery

Multicast routing protocols such as PIM-SM (Protocol Independent Multicast Sparse-Mode) and PIM-DM (Protocol Independent Multicast Dense-Mode) build the multicast delivery tree by using the RPF (Reverse-Path Forwarding) information toward the root of the tree. The root could be the so-called Rendezvous Point (RP) (in case of PIM-SM) or the source itself (in case of PIM-SM or PIM-DM).

The RPF information in each router is per multicast distribution tree and is basically the next-hop neighbor router information toward the root of the tree. In other words, the RPF router is the next-hop router toward the root. In case of PIM-SM, the RPF neighbor is typically the router that a Join message is sent to.

Obviously, all multicast routers must have consistent RPF state, otherwise a Join message might never reach the root of the tree. Typically, the unicast path forwarding information is used to create the RPF information, because under normal circumstances the unicast routing provides the necessary information to all routers.

Note that the unicast-based RPF creates multicast distribution trees where each branch of the tree follows the unicast path from each leaf of the tree toward the root. Usually this is the desired behavior, but occasionally someone may want the unicast and the multicast traffic to use different paths. For example, if a site has two links to its network provider, one of the links may be used for unicast only, and the other one only for multicast.

To provide for such flexibility in the configuration, the PIM-SM and PIM-DM specifications use the Multicast Routing Information Base (MRIB) for obtaining the RPF information. Typically, the MRIB may be derived from the unicast routing table, but some protocols such as MBGP may carry multicast-specific topology information. Furthermore, the MRIB may be modified locally in each site by taking into account local configuration and preferences. A secondary function of the MRIB is to provide routing metrics for destination addresses. Those metrics are used by the PIM-SM and PIM-DM Assert mechanism.

Configuring the MRIB

The Vyatta RIB module contains a table with the MRIB. That table is propagated to the PIM-SM module and is used by PIM-SM in the RPF computation. The MRIB table inside the RIB module is completely independent from the Unicast Routing Information Base (URIB) table. The URIB table is created from the unicast routes calculated by unicast routing protocols such as BGP, OSPF and RIP. The MRIB table is created similarly, but only by those protocols that are explicitly configured to add their routes to the MRIB.

For example, if Multi-protocol BGP is enabled, then the BGP module will add multicast-specific routes to the MRIB.

Currently, Vyatta supports the following methods for adding routing entries to the MRIB:

- Multi-protocol BGP.

The BGP module can be configured to negotiate multiprotocol support with its peers. Then, the BGP multicast routes will be installed in the MRIB. See “Chapter 10: BGP” for information how to configure BGP.

- Static Routes

  The Static Routes module can be used to configure static unicast and multicast routes. The unicast routes are added to the unicast RIB, while the multicast routes are added to the MRIB. See “Chapter 7: Static Routes” for information how to configure static routes.

- FIB2MRIB

  If there are no unicast routing protocols configured in the router to supply the MRIB routes, then the FIB2MRIB module can be used to populate the MRIB. If the FIB2MRIB module is enabled, it will register with the FEA to read the whole unicast forwarding table from the underlying system, and to receive notifications for all future modifications of that table. In other words, the FIB2MRIB’s task is to replicate the unicast forwarding information on that router into the MRIB.

Route Selection Process

A router can run multiple routing protocols simultaneously. For example, you might use RIP to distribute routes within your network, but BGP to learn external routes. In some situations, this can lead to a router learning the same route from more than one routing protocol. For example, the router might learn the following two routes:

- Subnet: 128.16.64.0/24, nexthop: 192.150.187.1, learned from BGP via an external peering. AS Path: 123 567 987.
- Subnet: 128.16.64.0/24, nexthop: 10.0.0.2, learned from RIP with metric 13

The problem for the router is to choose the best of these two routes for the given address.

The longest prefix match rule does not help the router choose between these two routes, because the prefix lengths are the same and the metric used for RIP is not directly comparable against the AS path length or any other attribute attached to a BGP route.

the Vyatta system uses the concept of *administrative distance* to determine which route is the best—that is, which route “wins.” Essentially, each routing protocol is configured with a “distance.” If a route is learned from two protocols, then the version with the smallest distance wins.

The Vyatta router uses a predefined set of administrative distances. Currently, these are not configurable. These are as follows.

Table 5-1 Vyatta router predefined administrative distances

Type of Route	Distance
Directly connected subnets	0
Static routes	1
BGP, heard from external peer	20
OSPF	110
RIP	120
BGP, heard from internal peer	200
FIB2MRIB routes (Vyatta-specific, in MRIB only)	254

Configuring Forwarding

On the Vyatta system, unicast packet forwarding is enabled by default when the unicast protocol configuration node is created. Multicast packet forwarding must be explicitly enabled.

Example 5-1 enables multicast forwarding on interface eth0, which permits IGMP (a multicast protocol) to be configured on that interface.

Example 5-1 Enabling multicast forwarding

```
vyatta@R1# set multicast mfea4 interface eth0
[edit]
vyatta@R1# set protocols igmp interface eth0
[edit]
vyatta@R1# show igmp interface
Interface    State    Querier         Timeout Version Groups
eth0         UP       10.1.0.100         None       2      3
eth1         DISABLED 0.0.0.0            None       2      0
eth2         DISABLED 0.0.0.0            None       2      0
lo           DISABLED 0.0.0.0            None       2      0
vyatta@R1# show mfea interface
Interface    State    Vif/PifIndex Addr            Flags
eth0         UP                0/2 10.1.0.100     MULTICAST BROADCAST KERN_UP
```

```
eth1          DISABLED          1/3                 MULTICAST BROADCAST
eth2          DISABLED          2/4                 MULTICAST BROADCAST KERN_UP
lo            DISABLED          3/1                 LOOPBACK KERN_UP
register_vif DISABLED           4/2 10.1.0.100      PIM_REGISTER KERN_UP
```

Displaying Route Information

Example 5-2 shows all routes in the RIB using the default output format (brief).

Example 5-2 “show route”: Displaying routes

```
vyatta@vyatta> show route
Total routes: 13, Total paths: 13
10.0.0.0/8      [static(1)]     > to 192.168.2.1  via eth2/eth2
10.0.0.0/24     [connected(0)]  > to 10.0.0.50  via eth0/eth0
25.0.0.0/8      [ebgp(0)]       > to 10.0.0.100 via eth0/eth0
25.25.0.0/16    [ebgp(0)]       > to 10.0.0.100 via eth0/eth0
25.25.25.0/24   [ebgp(0)]       > to 10.0.0.100   via eth0/eth0
26.0.0.0/8      [ospf(1)]       > to 10.0.0.100 via eth0/eth0
26.26.0.0/16    [ospf(1)]       > to 10.0.0.100 via eth0/eth0
26.26.26.0/24   [ospf(1)]       > to 10.0.0.100 via eth0/eth0
27.0.0.0/8      [rip(2)]        > to 10.0.0.100 via eth0/eth0
27.27.0.0/16    [rip(2)]        > to 10.0.0.100 via eth0/eth0
27.27.27.0/24   [rip(2)]        > to 10.0.0.100 via eth0/eth0
172.16.0.0/14   [connected(0)]  > to 172.16.0.50  via eth1/eth1
192.168.2.0/24  [connected(0)]  > to 192.168.2.31 via eth2/eth2
```

The Vyatta system returns the longest matching unique dotted decimal entry for the specified prefix, even if the entry does not exactly match the dotted decimal subnet. I the entry does not match, only the closest (longest) matching prefix is returned.

Example 5-3 shows the longest prefix matching mechanism.

Example 5-3 “show route”: Longest prefix matching

```
vyatta@R1> show route
Total routes: 9, Total paths: 9
0.0.0.0/0               [static(1)]     > to 10.0.0.1           via eth0
10.0.0.0/24             [connected(0)]  > to 10.0.0.234         via eth0
172.0.0.0/8             [static(1)]     > to 10.0.0.1           via eth0
172.16.0.0/16           [static(1)]     > to 10.0.0.1           via eth0
172.16.0.0/24           [static(1)]     > to 10.0.0.1           via eth0
172.16.0.0/25           [static(1)]     > to 10.0.0.1           via eth0
```

```
172.17.0.0/16           [static(1)]     > to 10.0.0.1           via eth0
172.32.0.0/24           [static(1)]     > to 10.0.0.1           via eth0
172.64.0.0/25           [static(1)]     > to 10.0.0.1           via eth0
vyatta@R1> show route 172.16.0.255
Total routes: 1, Total paths: 1
172.16.0.0/24           [static(1)]     > to 10.0.0.1           via eth0
```

Example 5-4 displays static routes.

Example 5-4 "show route": Displaying static routes

```
vyatta@vyatta> show route protocol static
Total routes: 13, Total paths: 13
Routes in this view: 1, Paths in this view: 1

10.0.0.0/8      [static(1)]     > to 192.168.2.1  via eth2/eth2
```

Example 5-5 displays routes with a prefix length of 16.

Example 5-5 "show route": Displaying routes of a specified prefix length

```
vyatta@vyatta> show route prefix-length 16
Total routes: 13, Total paths: 13
Routes in this view: 2, Paths in this view: 2

25.25.0.0/16    [ebgp(0)]       > to 10.0.0.100 via eth0/eth0
26.26.0.0/16    [ospf(1)]       > to 10.0.0.100 via eth0/eth0
27.27.0.0/16    [rip(2)]        > to 10.0.0.100 via eth0/eth0
```

Example 5-6 displays routes with a next hop of 10.0.0.100.

Example 5-6 "show route": Displaying routes with a specified next hop

```
vyatta@vyatta> show route next-hop 10.0.0.100
Total routes: 13, Total paths: 13
Routes in this view: 9, Paths in this view: 9

25.0.0.0/8      [ebgp(0)]       > to 10.0.0.100 via eth0/eth0
25.25.0.0/16    [ebgp(0)]       > to 10.0.0.100 via eth0/eth0
25.25.25.0/24   [ebgp(0)]      > to 10.0.0.100   via eth0/eth0
26.0.0.0/8      [ospf(1)]       > to 10.0.0.100 via eth0/eth0
26.26.0.0/16    [ospf(1)]       > to 10.0.0.100 via eth0/eth0
```

```
26.26.26.0/24   [ospf(1)]       > to 10.0.0.100 via eth0/eth0
27.0.0.0/8      [rip(2)]        > to 10.0.0.100 via eth0/eth0
27.27.0.0/16    [rip(2)]        > to 10.0.0.100 via eth0/eth0
27.27.27.0/24   [rip(2)]        > to 10.0.0.100 via eth0/eth0
```

Chapter 6: Routing Policies

This chapter describes the policy framework, which you can use to apply policies that influence routing behavior.

The following topics are presented:

- Policy Overview
- Policy Configuration
- Policy Objects
- Policy Evaluation
- Applying Policy Statements to Routing Protocols
- Specifying Match Criteria Operators
- Regular Expressions
- Policy Configuration Examples

Policy Overview

A policy statement is a mechanism that allows a user to configure criteria to compare a route against an action(s) to be performed on the route if the defined criteria are met. For example, a policy statement can be used to filter (block) specific route prefixes that are being announced by a BGP neighbor. Policy statements are also used to export routes learned via one protocol, for instance OSPF, into another protocol, for instance BGP. This is commonly called *route redistribution*.

Policy statements are grouped together in the Vyatta configuration under the **policy** node. This **"policy"** node simply serves as a container for policy statements; it's the actual policy statements that define the rules that will be applied to routing updates.

Once a policy statement has been defined, in order for the policy statement to take affect, the policy statement needs to be applied to a specific routing protocol. Policy statements can be applied as either an *import* policy or an *export* policy to routing protocols like OSPF and BGP.

Policy statements that have been applied as an *import* policy to a routing protocol are used to evaluate routing updates *received* via the routing protocol to which the policy statement is applied. For example, if a user configures an import policy for BGP, all BGP announcements received by the Vyatta router will be compared against the import policy statement first, prior to being added to the BGP and routing tables.

Policy statements that have been applied as an *export* policy to a routing protocol are used to evaluate routing updates that are *transmitted* by the routing protocol to which the policy is applied. For example, if a user configures an export policy for BGP, all BGP updates originated by the Vyatta router will be compared against the export policy statement prior to the routing updates being sent to any BGP peers.

In addition to controlling routing updates transmitted by a routing protocol, export policies are also used to provide route redistribution. For example, if a user wants to redistribute routes learned via OSPF into BGP, the user would configure a policy statement identifying the OSPF routes of interest, and then the user would apply this policy statement as an export policy for BGP.

Policy Configuration

This section presents the following topics:

- Building a Policy Statement
- Policy Configuration Options
- Policy Component Interaction

Policies are configured in the CLI using the **set policy policy-statement...** command syntax. Each **policy-statement** must contain at least one term, and within a term there are two types of match statements and one type of action statement.

- The match statements use the configuration keywords **from** and **to**.
- The action statement uses the configuration keyword **then**.

Once a policy statement has been configured, it must be applied to a specific routing protocol to have any effect. In this section, we'll describe the configuration of policy statements.

The section "Applying Policy Statements to Routing Protocols" on page 72 describes how policy statements are applied to protocols to control routing behavior. The section "Policy Configuration Examples" on page 80 show examples of applying policies to routing protocols.

The examples in this section lay out the configuration structure for defining policy statements. These examples are shown in "configuration mode" syntax. Each example configuration section builds on the previous example, showing how policy statements can be built from the "top down." User-defined values are shown in blue text.

Building a Policy Statement

The **policy** configuration node is a container that can hold multiple **policy-statements**. It is the actual policy statements that are applied to a routing protocol.

Example 6-1 shows a policy containing two empty policy statements.

Example 6-1 Empty policy statements

```
policy {
   policy-statement POLICY-NAME-1 {
   }
   policy-statement POLICY-NAME-2 {
   }
}
```

A policy statement has a text name that is used to reference the policy statement in other configuration sections. This name is represented in the example above by the text in italic.

Each **policy-statement** node can contain multiple **term** nodes which are evaluated in the sequence in which they are entered into the configuration. Example 6-2 shows two **policy-statement** nodes, each with two empty **term** statements.

Example 6-2 Policy statement with empty term statements

```
policy {
     policy-statement POLICY-NAME-1 {
            term TERM-NAME-1 {
            }
            term TERM-NAME-2 {
            }
     }
     policy-statement POLICY-NAME-2 {
            term TERM-NAME-1 {
            }
            term TERM-NAME-2 {
            }
     }
}
```

A term has a text name that is used to differentiate it from other terms in the same policy statement. This name is frequently an integer number, like 1, but it does not have to be.

Each **term** node can contain match statements and an action statement. The match statements are defined by the **from** and **to** configuration keywords. The action statement is defined by the **then** configuration keyword.

Example 6-3 shows a single policy statement (the second policy statement has been removed for brevity), with empty match statements and empty action statements.

Example 6-3 Policy statement with empty match and action statements

```
policy {
   policy-statement POLICY-NAME-1 {
      term TERM-NAME-1 {
         from {
         }
         to {
         }
         then {
         }
      }
      term TERM-NAME-2 {
         from {
         }
         to {
         }
         then {
         }
      }
```

```
    }
}
```

If defined, the match statements must include at least one match criterion. The match criteria are a predefined list of configuration keywords, such as **network4**, that are used to define the attributes of the routing entry the user desires to match.

The action statement must include at least one action directive. The action directives are also a predefined list of configuration keywords.

The configuration below shows a single policy-statement with a single “term” defined. As you can see, each match statement and action statement can have more than one entry

Example 6-4 Policy statement with a term defined

```
policy {
    policy-statement POLICY-NAME-1 {
        term TERM-NAME-1 {
            from {
                MATCH-CRITERION-1
                MATCH-CRITERION-2
            }
            to {
                MATCH-CRITERION-3
                MATCH-CRITERION-4
            }
            then {
                ACTION-DIRECTIVE-1
                ACTION-DIRECTIVE-2
            }
        }
    }
}
```

If all the matching criteria in the match statements are matched against the route entry, then the actions defined in the action statement are taken. If match criteria are entered in both the **from** and **to** sections of the term, then all match criteria in both the **from** and **to** section must match before the defined action statement is performed.

So far all the examples have been shown in “configuration node” syntax. In order to generate a policy configuration, the user must enter the appropriate CLI commands. Example 6-5 shows CLI command examples which demonstrate how to create a policy statement with **from** and **to** match statements and an action statement. The text in blue shows the user-defined values.

Example 6-5 Policy statement configuration CLI commands

```
set policy policy-statement EXAMPLE term 1 from network4
10.0.0.0/8
set policy policy-statement EXAMPLE term 1 to nexthop4 10.0.0.1
set policy policy-statement EXAMPLE term 1 then action accept
```

These commands would result in the configuration node structure shown in Example 6-6; that is, this is what would be shown if the user typed **show** while in configuration mode.

Example 6-6 Configuration node structure resulting from CLI commands

```
policy {
   policy-statement EXAMPLE {
      term 1 {
         from {
            network4: 10.0.0.0/8
         }
         to {
            nexthop4: 10.0.0.1
         }
         then {
            action: accept
         }
      }
   }
}
```

Policy Configuration Options

This section provides a brief description of the various options that can be used in policy statements. The following options are described:

- Policy-Statements
- Terms
- Match Statements
- Match Criteria
- Operators
- Action Statements

Policy-Statements

Policy statements are defined by a *POLICY-NAME* field that creates a text identifier. This identifier can be used to reference the policy in other configuration sections, such as routing protocols.

Terms

Every policy must contain at least one term, but typically a policy will have multiple terms. The *TERM-NAME* field is a text field that identifies a specific term. The term contains a grouping that defines a set of criteria together with a set of actions to be taken if all the criteria are met.

A common practice is to use an integer for the *TERM-NAME*; however, terms are evaluated in the sequence in which they are entered into the configuration, not by the term value.

Match Statements

The match statement contains the match criteria defined by the **from** and **to** configuration nodes. In order for a route to be considered a match, all match criteria defined in the **from** and **to** components must match.

For policies that are applied as an *import* policy, if no match statements (that is, no **from** or **to** components) are defined, then *all* route prefixes will match the term and the action statement associated with the policy term will be applied.

Currently *export* policies require a minimum policy term configuration that includes a match criteria identifying the source protocol using the **from protocol …** configuration node. If no other match criteria are defined, the export policy will match *all* routes originating from the source protocol, and the action statement associated with the policy term will be applied.

Match Criteria

Match criteria are used in combination with the **from** and **to** match statements to define the criteria that should be used to identify a specific route that the policy term is attempting to match. For example, a user could specify a term with the match statement **from network4 10.0.0.0/8**, where the **network4** is the match criterion and **10.0.0.0/8** is the match value. If more than one match criteria is defined, the route must match *all* the criteria in order for the defined action to be taken.

Operators

Some match criteria can use operators in addition to the criteria value. For example, a **from** policy statement term could include a **prefix-length4 > 24** statement. This would match routes with a prefix length greater than 24, where the “>” sign is the operator.

Action Statements

Action statements are defined by the configuration node **then**. These identify what action(s) should be performed if all the match criteria for a term are matched. An action could be as simple as accepting a route, or it could include performing activities like changing the next hop of the routing update.

For policy statements applied as an *import* policy, there is an implied **accept** action at the end of the policy statement. For example, if a user configured an import policy with a single term to reject a specific route, then that specific route will be filtered out, but all other routes will be accepted by default.

Policy statements applied as an *export* policy do not have an implied action at the end of the policy statement. For example, if a user configures an export policy with a single term to allow a specific route to be announced, then all other routes which do not match the match criteria will not be announced.

Policy Component Interaction

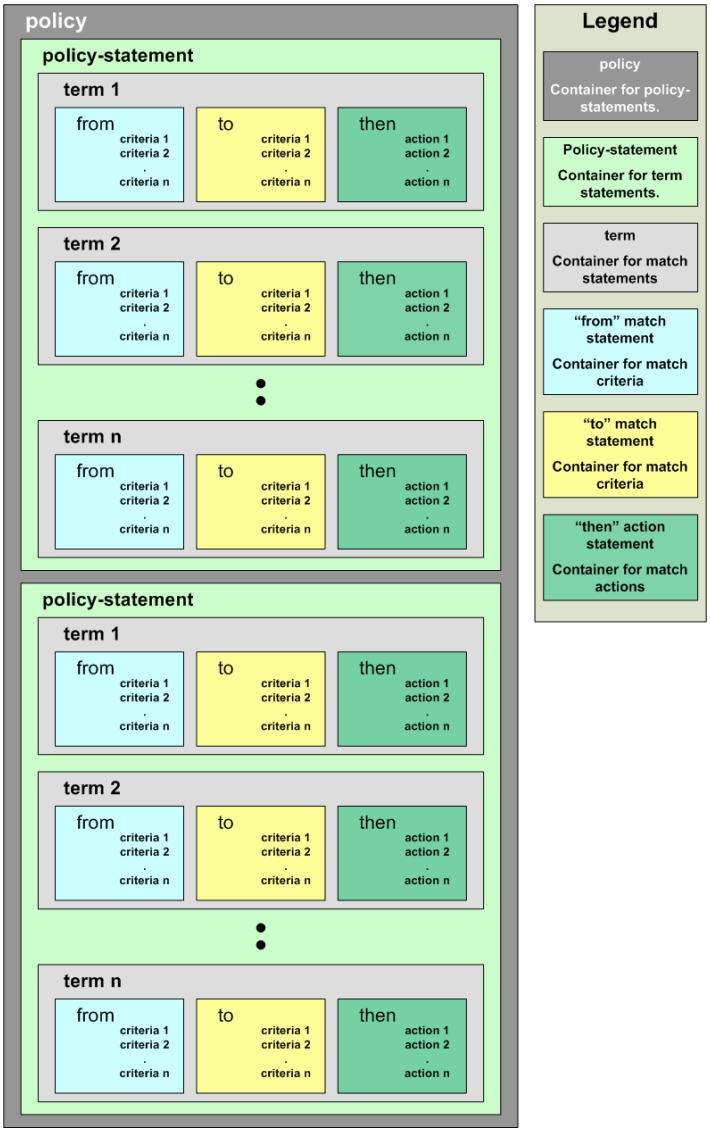
All policy statement terms consist of up to two match statements and one action statement.

- **from**. The **from** match statement in a term contains one or more match criteria for information about the source of the route; for example, **from network4** with a specific IP network of the route.
- **to**. The **to** match statement in a term contains one or more match criteria for information about the destination of the route; for example, **to nexthop4** with a specific IP address of the next hop that the route is being sent to.
- **then**. The **then** action statement in a term defines one or more actions that will be taken if all match criteria are met. The action can be to accept the route (**then action accept**), to reject the route (**then action reject**), or to perform some other action such as specifying the next hop (**then nexthop4** *ip-addr*) or setting the metric (**then metric** *metric*) For example, accepting the route would direct the Vyatta router to allow the route to be received for an import policy, or indicate that the route should be announced for an export policy.

Note that each match statement can contain one or more match criteria, and each action statement can contain one or more actions.

Figure 6-1 shows how each of the policy configuration components fit together.

Figure 6-1 Interaction of policy components



Policy Objects

This section presents the following topics:

- Supported Policy Objects
- Configuring Policy Objects
- Defining a BGP AS Path List
- Defining a BGP Community List

In order to help reduce the number of terms needed to implement a policy, the Vyatta router supports policy objects. Policy objects enable a user to create a grouping of items, like IP subnets, that should all have the same policy action taken for matching networks. For example, policy objects allow a user to create a single policy term containing a list of networks that should be accepted from a BGP neighbor.

Supported Policy Objects

Table 6-1 lists the policy objects that are currently supported.

Table 6-1 Supported Policy Objects

Policy Object	CLI Example
as-path-list	set policy as-path-list MY-PATHS elements “1 2,1 5,2 7"
community-list	set policy community-list MY-COMMS elements “comm-1,comm-2”
network4-list	set policy network4-list MY-NETS elements “1.0.0.0/8,2.0.0.0/8”
network6-list	set policy network6-list MY-IPV6-NETS elements “

Configuring Policy Objects

Configuring a policy object results in the following configuration structure, where user-defined values are shown in blue text.

Example 6-7 Configured policy object

```
policy {
   network4-list <LIST-NAME> {
      elements: "1.0.0.0/8,2.0.0.0/8"
   }
```

In order to generate a policy object configuration, the user must enter the appropriate CLI commands. Example 6-5 shows CLI command examples which demonstrate how to create a policy object with two IP subnets.

Example 6-8 Policy object with two IP subnets

```
set policy network4-list MY-NETS elements "1.0.0.0/8,2.0.0.0/8"
```

Once a policy object has been created, it can be referenced within a policy **term** statement. The following configuration example shows an import policy that rejects the IP networks 1.0.0.0/8 and 2.0.0.0/8. The user-defined values are shown in blue text.

Example 6-9 Import policy rejecting two IP networks

```
policy {
   policy-statement POLICY-1 {
      term 1 {
         from {
            network4-list: "MY-NETS"
         }
         then {
            action reject
         }
      }
   }
}
```

Defining a BGP AS Path List

An AS path is a list of the ASs that a routing update traversed to a destination in the Border Gateway Protocol (BGP). The path is represented as a sequence of AS numbers, which are the numbers uniquely identifying BGP autonomous systems. Each AS number represents a network that a packet traverses if it takes the associated route to the destination.

For a packet to reach a destination using this route, it traverses the listed ASs from the leftmost AS number to the rightmost, where the rightmost is the AS immediately preceding its destination, also known as the “origin” AS.

Using policies, match conditions can be defined based on all or portions of the AS path. To do this, you create a named AS path regular expression and then include it in a routing policy.

The AS can be specified in either of the following ways:

- Using the AS number. The AS number as a whole counts as a single term. That is, you cannot reference individual characters within an AS number.
- Using a group of AS numbers enclosed in double quotes. When you group AS numbers, the operator acts on the entire group. The grouped path itself can include operators.

For example, the following AS path list:

Example 6-10 AS path list

```
“3 2 1”
```

matches all AS paths using this exact AS sequence, and no others. It does not match, for instance the AS sequence **“4 3 2 1”**.

You can use wildcards from Table 6-5 to create more powerful AS path list specifications. For example, to match **“*n* 3 2 1”**, where *n* is any AS, you could use the following AS path list:

Example 6-11 AS path list using “?” wildcard

```
“? 3 2 1”
```

This list matches all of the following:

```
“4 3 2 1”
“100 3 2 1”
“55 3 2 1”
```

To match **“***list* **3 2 1”**, where *list* is any list of AS numbers, you could use the following community list:

Example 6-12 AS path list using “*” wildcard

```
“* 3 2 1”
```

This list matches all of the following:

```
“4 3 2 1”
“100 3 2 1”
“55 3 2 1”
“4 55 3 2 1”
“4 100 55 3 2 1”
```

Defining a BGP Community List

All BGP updates include a BGP attribute called the communities path attribute. The communities path attribute allows ASs to “tag” prefix announcements. This tag can then be used by routing policies to modify the normal behavior for that prefix announcement. For example, an AS could choose to filter out all prefix announcements containing a community value that identifies the prefix as a customer-generated prefix, instead only announcing the summary prefix for all customer prefixes. It is important to note that the community path attribute is carried in BGP update messages, which allows ASs not directly connected to each other to share information about a prefix.

The format for community identifiers is defined in RFC 1997: “BGP Communities Attribute.” The community identifier is a 32-bit value, where the first two bytes of the value are the AS number and the second two bytes are an arbitrary value defined by the AS. This format can be represented as *AA:NN*, where *AA* is the AS number of the AS adding the community identifier to the community path attribute, and *NN* represents a user-defined policy value.

There are two types of BGP communities: “well-known” communities and user-defined or private communities.

The Vyatta router recognizes the following BGP well-known communities as per RFC 1997:

NO_EXPORT: All routes received carrying a communities attribute containing this value are not advertised outside a BGP confederation boundary (a stand-alone autonomous system that is not part of a confederation should be considered a confederation itself).

NO_ADVERTISE: All routes received carrying a communities attribute containing this value are not advertised to other BGP peers.

NO_EXPORT_SUBCONFED: All routes received carrying a communities attribute containing this value are not advertised to external BGP peers (this includes peers in other members autonomous systems inside a BGP confederation).

Policy Evaluation

Once a policy statement has been applied as an import or export policy for a routing protocol, the terms within the policy statement are evaluated in the order that they are entered into the configuration. Policy statements are evaluated term by term. A then accept or a then reject causes an exit on the first successful match; otherwise, the route is sent to the next term.

XORP Docs state that a then accept or then reject will lead to an exit on first match, otherwise the route is sent to the next term - this appears to be the case as seen in policy testing - you do say this toward the end of this document

If the route entries being evaluated do not exactly match any of the match criteria defined in the match statements for any of the terms in the policy statement, the default policy action for import policies is to **accept** the route. For export policies, only those routes that match a policy term will be acted on; all other routes will be ignored by default, resulting in an implied **reject all** as the last term of an export policy.

Applying Policy Statements to Routing Protocols

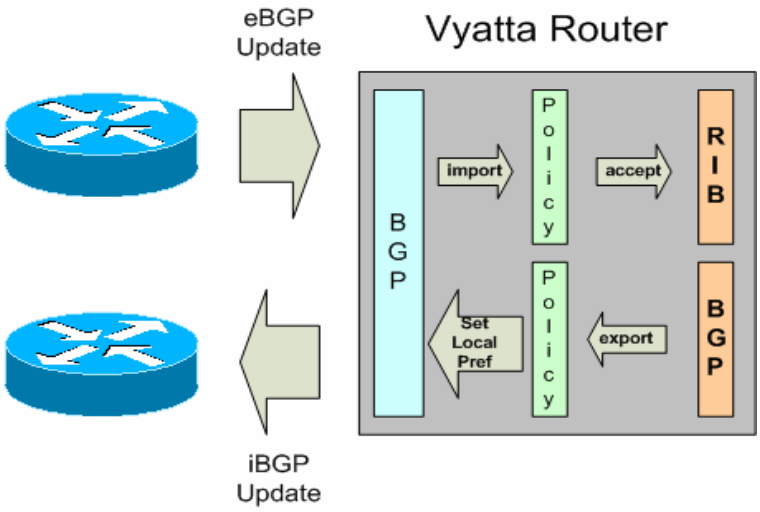
After a policy statement has been defined, the policy will not have any impact on routing updates until it is applied to a routing protocol as an import or export policy.

Import routing policies control the behavior of routing updates that are received by a routing protocol. For example, if a user wanted to change the metric of routes learned via OSPF from its OSPF neighbors, the user would configure an OSPF import policy.

Export routing policies control the behavior of routing updates that are sent by a routing protocol. For example, if a user wanted to filter out a specific prefix from being sent to BGP neighbors, the user would configure a BGP export policy. Export routing policies are also used to perform route redistribution from one protocol to another.

Figure 6-2 shows the behavior for BGP when both an import and export policy statement is configured. In this example, the import policy is filtering for allowed prefixes that the Vyatta router should accept from its BGP neighbors. The export policy is setting the Local Pref for updates sent to its BGP neighbors.

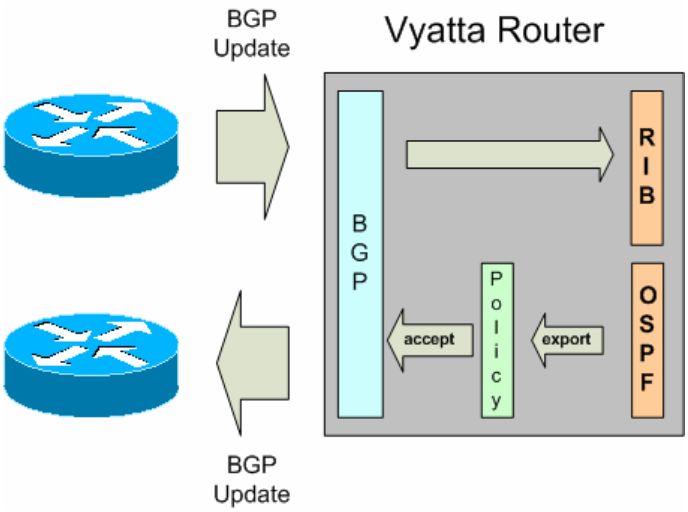
Figure 6-2 BGP with an import and an export policy statement



All export policy terms require a defined **from protocol** field that specifies the source protocol. If the source protocol defined in **from protocol …** is the same as the protocol that the policy statement is applied to as an export statement, then the routes from that protocol will flow through the policy statement and have the appropriate actions taken. This behavior is shown in Figure 6-2.

If the source protocol defined in the **from protocol …** is different from the protocol that the policy statement is applied to as an export statement, then the routes from the source protocol will be redistributed into the export protocol. For example, to redistribute OSPF routes into BGP, the source protocol defined in the policy statement would be **from protocol ospf**, and the policy statement would be applied as an export policy in BGP. Figure 6-3 shows this behavior.

Figure 6-3 Redistributing routes from OSPF into BGP



Once a policy statement has been defined, it must be explicitly applied to the routing protocol using the import and/or export directives in routing protocol configuration.

Not every policy criterion in the **from**, **to**, and **then** parts of the term can be applied to every routing protocol; the applicable options vary with the protocol. Table 6-2 shows which options apply to which protocols.

Table 6-2 Policy Options Applicable per Protocol

from	BGP	RIP	RIPng	OSPF	Static
protocol	×	×	×	×	×
network4	×	×	×	×	×
network6	×	×	×	×	×
network4-list	×	×	×	×	×
network6-list	×	×	×	×	×
prefix-length4	×	×	×	×	×
prefix-length6	×	×	×	×	×
nexthop4	×	×		×	
nexthop6	×		×		
as-path	×				
as-path-list	×				

Table 6-2 Policy Options Applicable per Protocol

community	×				
community-list	×				
neighbor	×				
origin	×				
med	×				
localpref	×				
metric		×	×	×	×
external				×	
tag		×	×	×	
to	**BGP**	**RIP**	**RIPng**	**OSPF**	**Static**
network4	×	×	×	×	×
network6	×	×	×	×	×
network4-list	×	×	×	×	×
network6-list	×	×	×	×	×
prefix-length4	×	×	×	×	×
prefix-length6	×	×	×	×	×
nexthop4	×	×		×	
nexthop6	×		×		
as-path	×				
as-path-list	×				
community	×				
community-list	×				
neighbor	×				
origin	×				
med	×				
localpref	×				
was-aggregated	×				
metric		×	×	×	

Table 6-2 Policy Options Applicable per Protocol

external				×	
tag		×	×	×	
then	BGP	RIP	RIPng	OSPF	Static
action	×	×	×	×	×
trace	×	×	×	×	×
nexthop4	×	×		×	
nexthop6	×		×		
as-path-prepend	×				
as-path-expand	×				
community	×				
community-add	×				
community-del	×				
origin	×				
med	×				
med-remove	×				
localpref	×				
aggregate-prefix-len	×				
aggregate-brief-mode	×				
metric		×	×	×	
external				×	
tag		×	×	×	

Specifying Match Criteria Operators

Some of the match criteria defined in **from** and **to** policy-statement terms can use operators in addition to the criteria value. For example, a **from** policy-statement term could include a **prefix-length4 > 24** statement. This would match routes with a prefix length greater than 24. In this case, the greater-than sign (“>”) is the operator.

If no operator is explicitly defined, each criterion has a default operator value. For example, by default, the operator for **prefix-length4** is equals ("==").

Table 6-3 shows the definitions for policy operators.

Table 6-3 Operator Definitions

Operator	Example	Description
:	10.10.35.0**:**10.10.35.254	Specifies a range of values, such as a range of numbers or IP addresses. Example: "Is an IPv4 address between 10.10.35.0 and 10.10.35.254, inclusive."
==	==15	Is equal to. Example: "is equal to 15."
!=	!=0	Is not equal to. Example: "Is not equal to 0."
<	<15	Is less than. Example: "Is less than 15."
>	>12	Is greater than. Example: "Is greater than 12."
<=	<=12	Is less than or equal to. Example: "Is less than or equal to 12."
>=	>=12	Is greater than or equal to. Example: "Is greater than or equal to 12."

The following criteria allow operators.

Table 6-4 Matching criteria allowing operators

Criterion	Matching Operators Allowed
localpref	all
med	all
metric	all
neighbor	all
network4	all
network4-list	all
nexthop	all
origin	all
prefix-length	all
tag	all

Example 6-13 shows a configuration example setting a match criterion that uses an operator. This policy would match all routes with a prefix length greater than or equal to **/24**.

Example 6-13 Match criterion using an operator

```
vyatta@R1# set policy policy-statement SAMPLE-POLICY term 1 from
prefix-length >= 24
```

Regular Expressions

Regular expressions provide the ability to perform pattern matching are used to parse data sets within AS path lists and community lists. In general, a regular expression takes the following form:

<*regex-term*><*operator*>

where <*regex-term*> is a string to be matched, and <*operator*> is one of the operators shown in Table 6-3.

Note that operators must occur immediately after <*regex-term*> with no intervening space, with the following exceptions:

- The vertical bar operator (“|”) and hyphen (“-”) operator, both of which are placed between two terms
- Parentheses, which enclose <*regex-term*>s.

Table 6-5 shows the regular expression operators supported in policy statements.

Table 6-5 Regular expression operators

Operator	Description
{*m*,*n*}	At least *m* and at most *n* repetitions of *regex-term.* Both *m* and *n* must be positive integers, and *m* must be smaller than *n*.
{*m*}	Exactly *m* repetitions of *regex-term*. *m* must be a positive integer.
{*m*,}	*m* or more repetitions of *regex-term*. *m* must be a positive integer.
*	Zero or more repetitions of *regex-term*. This is equivalent to {0,}.

Table 6-5 Regular expression operators

Operator	Description
+	One or more repetitions of *regex-term*. This is equivalent to {1,}.
?	Zero or one repetition of *regex-term*. This is equivalent to {0,1}.
|	One of the two *regex-term* on either side of the vertical bar.
-	Between a starting and ending range, inclusive.
^	Character at the beginning of an AS path regular expression. This character is added implicitly; therefore, the use of it is optional.
$	Character at the end of an AS path regular expression. This character is added implicitly; therefore, the use of it is optional.
()	A group of *regex-term*s that are enclosed in the parentheses. If enclosed in quotation marks with no intervening space (“()”), indicates a null. Intervening space between the parentheses and the *regex-term* is ignored.
[]	Set of characters. One character from the set can match. To specify the start and end of a range, use a hyphen (-).
^	NOT operator.

Example 6-14 shows a configuration example setting a match criterion that uses an operator.

Example 6-14 Match criterion using a regular expression

```
vyatta@R1# set policy policy-statement SAMPLE-POLICY term 21
from as "^1$"
```

Policy Configuration Examples

This section provides the following examples.

- Example 6-15 Redistributing static routes into BGP
- Example 6-16 Redistributing static routes into OSPF
- Example 6-17 Filtering routes in BGP
- Example 6-18 Storing network prefixes in a policy object
- Example 6-19 Redistributing directly connected routes into RIP

Redistributing Static Routes into BGP

Tip: *It's a good convention to distinguish user-defined names from system-defined names. In this example, the policy statement names are all capitals.*

Example 6-15 redistributes two specific static routes into BGP. This example creates the policy EXPORT_STATIC, which has two terms:

- Term 1 accepts packets from static routes on network 192.168.10.0/24.
- Term 2 accepts packets from static routes on network 192.168.20.0/24.

The policy is then applied to the BGP protocol. If these static routes exist in the RIB, then they will be announced via BGP.

To create this policy, perform the following steps in configuration mode:

Example 6-15 Redistributing static routes into BGP

Step	Command
Create the first term. To match, packets must be from a static route.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 1 from protocol static` `[edit]`
To match, packets must be from network 192.168.10.0/24.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 1 from network4 192.168.10.0/24` `[edit]`
If a packet matches both criteria, accept it.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 1 then action accept` `[edit]`
Create the second term. To match, packets must be from a static route.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 2 from protocol static` `[edit]`
To match, packets must be from network 192.168.20.0/24.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 2 from network4 192.168.20.0/24` `[edit]`

Example 6-15 Redistributing static routes into BGP

If a packet matches both criteria, accept it.	`vyatta@R1# set policy policy-statement EXPORT_STATIC term 2 then action accept` `[edit]`
Apply this policy to the BGP protocol.	`vyatta@R1# set protocols bgp export EXPORT_STATIC` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Redistributing Static Routes into OSPF

Example 6-16 creates the policy EXPORT_ALL_STATIC, which redistributes all static routes into OSPF. If these static routes exist in the RIB, then they will be announced via OSPF.

To create this policy, perform the following steps in configuration mode:

Example 6-16 Redistributing static routes into OSPF

Step	Command
Create the first term. Match packets from static routes.	`vyatta@R1# set policy policy-statement EXPORT_ALL_STATIC term 1 from protocol static` `[edit]`
If a packet matches, accept it.	`vyatta@R1# set policy policy-statement EXPORT_ALL_STATIC term 1 then action accept` `[edit]`
Apply this policy to the OSPF protocol.	`vyatta@R1# set protocols ospf export EXPORT_ALL_STATIC` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Filtering Routes in BGP

Example 6-17 creates the policy IMPORT_FILTER, which filters the routing updates from BGP neighbors that are allowed by the local router. This example allows two prefixes: 172.17.0.0/16 and 172.19.0.0/24. All other prefixes are rejected.

To create this policy, perform the following steps in configuration mode:

Example 6-17 Filtering routes in BGP

Step	Command
Create the first term. Match packets from network 172.17.0.0/16.	`vyatta@R1# set policy policy-statement IMPORT_FILTER term 1 from network4 172.17.0.0/16` `[edit]`
If a packet matches, accept it.	`vyatta@R1# set policy policy-statement IMPORT_FILTER term 1 then action accept` `[edit]`
Create the second term. Match packets from network 172.19.0.0/16.	`vyatta@R1# set policy policy-statement IMPORT_FILTER term 2 from network4 172.19.0.0/16` `[edit]`
If a packet matches, accept it.	`vyatta@R1# set policy policy-statement IMPORT_FILTER term 2 then action accept` `[edit]`
Create a third term. Reject any packets that have not been matched.	`vyatta@R1# set policy policy-statement IMPORT_FILTER term 3 then action reject` `[edit]`
Apply this policy to the BGP protocol.	`vyatta@R1# set protocols bgp import IMPORT_FILTER` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Storing Network Prefixes in a Policy Object

Example 6-18 is exactly the same as Example 6-17., except this example uses a policy object to store both prefixes in a single object.

To create this policy, perform the following steps in configuration mode:

Example 6-18 Storing network prefixes in a policy object

Step	Command
Create the policy object storing the list of networks.	`vyatta@R1# set policy network4-list NETWORK_OBJECT elements "172.17.0.0/16,172.19.0.0/16"` `[edit]`
Create the policy term using the network list.	`vyatta@R1# set policy policy-statement IMPORT_FILTER_2 term 1 from network4-list NETWORK_OBJECT` `[edit]`
If a packet matches, accept it.	`vyatta@R1# set policy policy-statement IMPORT_FILTER_2 term 1 then action accept` `[edit]`
Create a second term. Reject any packets that have not been matched.	`vyatta@R1# set policy policy-statement IMPORT_FILTER_2 term 2 then action reject` `[edit]`
Apply this policy to the BGP protocol.	`vyatta@R1# set protocols bgp import IMPORT_FILTER_2` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Redistributing Directly Connected Routes

In this release if you want a routing protocol, such as RIP, to announce connected interfaces (including those with RIP configured) you must define a policy for redistributing connected routes.

Note that once defined, the policy must be applied to specific routing protocols using the **import** or **export** directive within that protocol.

Example 6-19 creates the policy EXPORT_CONN. This policy directs the routing protocol to redistribute all directly connected routes.

To create a policy for redistributing directly connected routes, perform the following steps in configuration mode:

Example 6-19 Redistributing directly connected routes into RIP

Step	Command
Create the first term. Match all directly connected routes.	vyatta@R1# **set policy policy-statement EXPORT_CONN term 1 from protocol connected** [edit]
If a packet matches, accept it.	vyatta@R1# **set policy policy-statement EXPORT_CONN term 10 then action accept** [edit]
Create a second term. Reject any packets that have not been matched.	vyatta@R1# **set policy policy-statement EXPORT_CONN term 2 then action reject** [edit]
Apply this policy to the RIP protocol.	vyatta@R1# **set protocols rip import EXPORT_CONN** [edit]
Commit the configuration.	vyatta@R1# **commit** OK [edit]

Chapter 7: Static Routes

This chapter describes how to configure static routes on the Vyatta system.

The following topics are covered:

- Static Routes Overview
- Configuring Static Routes
- Monitoring Static Route Information

Static Routes Overview

A static route is a manually configured route, which in general cannot be updated dynamically from information the router learns about the network topology. However, if a link fails, the router will remove routes, including static routes, from the RIB that used that interface to reach the next hop.

In general, static routes should only be used for very simple network topologies, or to override the behavior of a dynamic routing protocol for a small number of routes.

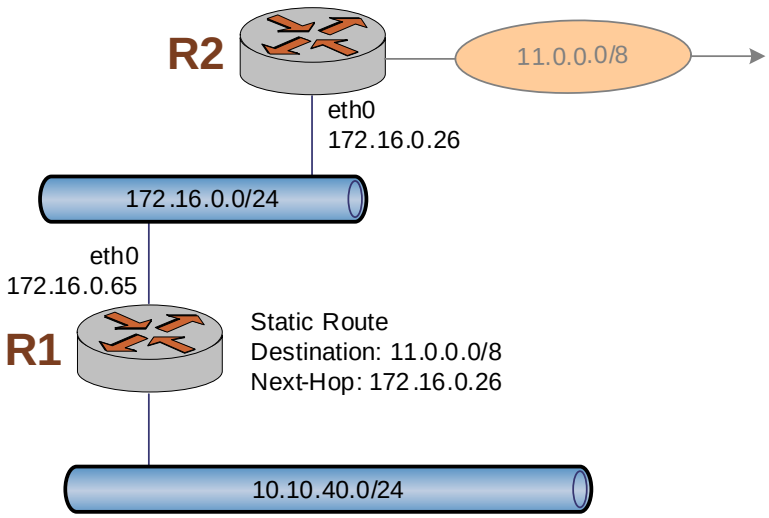
The collection of all routes the router has learned from its configuration or from its dynamic routing protocols is stored in its Routing Information Base (RIB).

Unicast routes are directly used to determine the forwarding table used for unicast packet forwarding.

Configuring Static Routes

In this section, sample configurations are presented for static routes. When you are finished, the router will be configured as shown in Figure 7-1.

Figure 7-1 Static routes



This section includes the following examples:

- Example 7-1 Creating a static route

Example 7-1 creates a static route to network 11.0.0.0/8 directed towards 172.16.0.26.

To create a static route, perform the following steps in configuration mode:

Example 7-1 Creating a static route

Step	Command
Create a mapping between the host name and the IP address.	`vyatta@R1# set protocols static route 11.0.0.0/8 next-hop 172.16.0.26` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Monitoring Static Route Information

This section presents the following topic:

- Static Route Operational Commands

Static Route Operational Commands

You can use the following operational commands to monitor static routes. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`show route`	Displays information about routes stored in the routing table.

This section presents the following examples:

- Example 7-2 Showing static routes in the routing table

Showing Static Routes in the Routing Table

To display route information, use the **show route** command. To show just static routes, use the **show route protocols static** filter, as shown in Example 7-2.

Example 7-2 Showing static routes in the routing table

```
vyatta@R1> show route protocol static
Routes: 2/9, Paths: 2/9
0.0.0.0/0               [static(1)]     > to 10.1.0.1           via eth0
10.7.0.48/28            [static(1)]     > to 10.6.0.57          via eth1

vyatta@R1>
```

Chapter 8: RIP

This chapter describes how to configure the Routing Information Protocol on the Vyatta system.

The following topics are covered:

- RIP Overview
- Supported Standards
- Configuring RIP
- Viewing RIP Information

RIP Overview

The Routing Information Protocol (RIP) is the simplest unicast routing protocol in widespread use today. RIP is very simple, both in configuration and protocol design, so it is widely used in simple topologies. However, RIP does not scale well to larger networks, where OSPF or IS-IS are generally more appropriate.

There have been two versions of the RIP protocol.

- RIP version 1 dates back to the early days of the Internet. It is now historic, primarily because it does not support classless addressing, which is necessary in today's Internet. The Vyatta system does not support RIPv1.
- RIP version 2 introduces a subnet mask, which allows classless addressing.The Vyatta system completely supports RIPv2, as specified in RFC 2453.

RIP is a distance vector protocol, which means that when a router receives a route from a neighbor, that route comes with a distance metric indicating the cost associated with reaching the destination via that neighbor. For RIP, this cost is the number of "hops" (that is, intermediate routers) to the destination.

The router adds its metric for the link on which the route was received to the metric in the received route, and then compares the route against its current best path to that destination. If the new route wins against all other factors, it is installed in the router's routing table. If the route is simply an update of the previous best route, then the stored metric is updated, and the route's deletion timer is restarted. Otherwise the route is ignored. Periodically, the router's routing table is sent to each of its neighbors. Additionally, if a route changes, then the new route is sent to each neighbor.

One reason why RIP is not good for large networks is that in complex topologies it is rather slow to conclude that a route is no longer usable. This is because routers in a loop will learn a route from each other all the way around the loop, and so when a destination becomes unreachable, the routing change will have to propagate around the loop multiple times, increasing the metric each time until the metric reaches infinity, when the route is finally removed. RIP uses a low value of 16 as infinity to reduce the time it takes to remove old information.

A simple case of such a loop is two routers talking to each other. After a destination becomes unreachable, two routers may each believe the other has the best route. *Split horizon* is a scheme for avoiding problems caused by including routes in updates sent to the router from which they were learned. The simple split horizon scheme omits routes learned from one neighbor in updates sent to that neighbor. *Split horizon with poisoned reverse* includes such routes in updates, but sets their metrics to infinity. In general, it is advisable to use split-horizon with poisoned reverse when using RIP, as this significantly speeds convergence in many scenarios.

Supported Standards

The Vyatta implementation of RIP complies with the following standard:

- RFC 2453: RIP version 2.

Configuring RIP

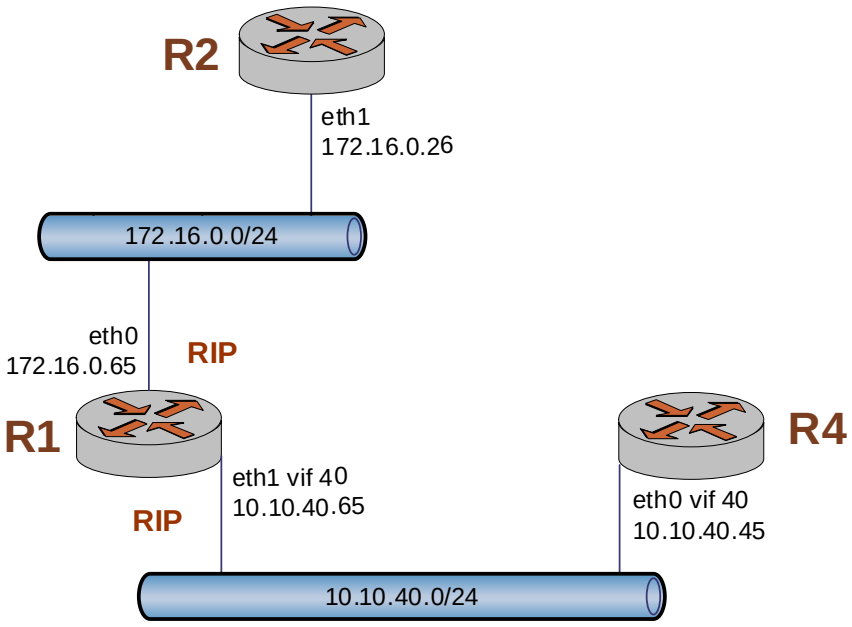
This section

To run RIP, you specify the set of interfaces, vifs and addresses on which RIP is to be enabled. Each address to be used by RIP must be explicitly added to the RIP configuration. Typically a metric will also be configured.

In addition, to originate routes via RIP, you must use the **export** parameter to export routes from the routing table via RIP. Also, unlike some other router implementations, on the Vyatta system you must export both static routes and connected routes into RIP in order announce these routes.

In this section, sample configurations are presented for RIP. The configuration used is shown in Figure 8-1.

Figure 8-1 RIP



This section includes the following examples:

- Example 8-1 Basic RIP Configuration

Basic RIP Configuration

Tip: *Note the notation for referring to vif 40 of interface eth1:* ***eth1.40****.*

Example 8-1 enables RIP on 172.16.0.65 on interface eth0 and 10.10.40.65 (on interface eth1.40.

To enable RIP, perform the following steps in configuration mode:

Example 8-1 Basic RIP Configuration

Step	Command
Create the RIP configuration node for 172.16.0.65. This enables RIP on that address on eth0.	`vyatta@R1# set protocols rip interface eth0 address 172.16.0.65` `[edit]`
Create the RIP configuration node for 10.10.40.65. This enables RIP on that address on eth1 vif 40.	`vyatta@R1# set protocols rip interface eth1.40 address 10.10.40.65` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Once RIP is configured, you must define policies to redistribute connected and static routes into RIP, as described in the next section.

Redistributing Static and Connected Routes into RIP

Directly connected routes must be explicitly redistributed by applying a routing policy using the **export** directive within RIP configuration. You can optionally also redistribute static routes.

“Chapter 6: Routing Policies” provides examples of routing policies applied to various routing protocols, including policies that redistribute static and connected routes. Please see that chapter for more information.

Viewing RIP Information

This section includes the following examples:

- Example 8-2 Showing RIP routes
- Example 8-3 Showing RIP peer information
- Example 8-4 Viewing the “protocols rip” configuration node

Showing RIP Routes

To display route information, use the **show route** command. To show just RIP routes, use the **show route protocol rip** option, as shown in Example 8-2.

Example 8-2 Showing RIP routes

```
vyatta@vyatta> show route protocol rip
```

Showing RIP Peers

To view information about RIP peers, use the **show rip peer** command in operational mode. Example 8-3 uses the **show rip peer statistics all** option of this command, showing RIP statistics for peer 172.16.0.26 on eth0 172.16.0.65.

Example 8-3 Showing RIP peer information

```
vyatta@R1> show rip peer statistics all
Last Active at Fri Jun  9 12:04:53 2006

  Counter                                 Value
  ------------------------------- ---------------
  Total Packets Received                  25966
  Request Packets Received                  167
  Update Packets Received                 25799
  Bad Packets Received                        0
  Authentication Failures                     0
  Bad Routes Received                         0
  Routes Active                               2
```

Showing RIP Configuration

You can always view the information in configuration nodes by using the **show** command in configuration mode. In this case, you can view RIP configuration by using the **show protocols rip command**, as shown in Example 8-4.

To show the information in the **protocols rip** configuration node, perform the following step in configuration mode:

Example 8-4 Viewing the “protocols rip” configuration node

Step	Command

Show the contents of the **protocols rip** configuration node.	`vyatta@R1# show protocols rip`
	`interface eth0 {`
	`address 172.16.0.65 {`
	`}`
	`}`
	`interface eth1.40 {`
	`address 10.10.40.65 {`
	`}`
	`}`
	`export: "EXPORT_CONNECTED,EXPORT_STATIC"`

Chapter 9: OSPF

This chapter describes how to configure the OSPF routing protocol on the router.

The following topics are covered:

- OSPF Overview
- Configuring OSPF
- Monitoring OSPF

OSPF Overview

The Open Shortest Path First (OSPF) routing protocol uses a link state algorithm (Dijkstra), as opposed to protocols such as RIP that use a distance vector algorithm. In OSPF, each router advertises the state of its own links, or connections, in a link state advertisement (LSA). It then multicasts the LSA to other routers on the network.

In addition, each router uses the LSAs it receives from other routers to construct a graph that represents the network topology. To build its routing table, the router applies Dijkstra's Shortest Path First algorithm to find the best path through the graph to each network in the topology. This "shortest path tree" becomes the basis for the routes that are elected by OSPF for inclusion in the routing table.

OSPF Areas

OSPF is hierarchical. In OSPF, the network is broken up into "areas." Within each area, routers possess only local routing information. Routing information about other areas is calculated using routes exchanged between areas. This reduces the amount of network topology information routers have to generate and maintain, making OSPF a better choice for larger networks.

No interface can belong to more than one area, and you should take care to avoid assigning overlapping address ranges for different areas. If all the interfaces on a router belong to the same area, the router is said to be an internal router. If the router has OSPF-enabled interfaces that belong to more than one area, it is said to be an Area Border Router (ABR). If a router imports routes from another protocol into OSPF, the router is said to be an Autonomous System Boundary Router (ASBR).

Note that an Area Border Router does not automatically summarize routes between areas. To summarize routes, you must explicitly configure router summarization using the **area-range** command.

An Area Border Router must be connected to the backbone area (0.0.0.0).

Specifying OSPF Areas

OSPF areas are defined by the interfaces that belong to them. Areas are defined in an IPv4 format and interfaces are configured to reside in a particular area.

```
vyatta@R1# set protocols ospf4 area 0.0.0.0 interface eth0
address 172.16.0.65
```

OSPF Area Types

This section discusses four types of OSPF areas:

- Normal Areas
- Stub Areas
- Not-So-Stubby-Areas
- The Backbone Area

Normal Areas

A normal area is an OSPF area that is not “special” in any way. It is not the backbone area, it is not a stub area, and it is not a not-so-stubby area (NSSA).

Stub Areas

A stub area is an OSPF area where no external link-state advertisements (type 5 LSAs) are allowed. A stub area contains an Area Border Router (ABR), but never contains an Autonomous System Boundary Router (ASBR). Therefore, from a stub area, destinations external to the AS can only be reached using the default summary route.

Stub areas are useful when a large part of the topology database consists of AS external advertisements. Routers in a stub area maintain a link state database that represents local links but no external links (type 4 or type 5 LSAs). Instead, external links are summarized into a default route and advertised using a single type 3 LSA generated by the ABR. This greatly reduces the size of the topology database. To replace externally-advertised routes, a stub area defines a default external route, called a summary route, through which traffic can reach external destinations.

You cannot configure an area as being both a stub area and an NSSA.

Not-So-Stubby-Areas

A Not-So-Stubby Area (NSSA) has some characteristics of a stub area, and some of a normal area. NSSAs are similar to stub areas in that they have only default routes to Autonomous System Boundary Routers (ASBRs). However, NSSAs can contain an ASBR (although they need not), while stub areas cannot.

You can configure an NSSA when you want to control the number of external routes in an area, but also allow an ASBR, or where you want OSPF traffic to be able to transit through.

Configuring an area as an NSSA allows a certain limited number of external routes to be redistributed into the network. Normally, ASBRs generate type 5 link-state advertisements (LSAs) for external routes, which advertise destinations external to the OSPF network

being redistributed into OSPF, and which are flooded to the entire autonomous system (AS). NSSAs generate Type 7 LSAs for flooding within the NSSA. These are then translated into Type 5 LSAs by the ABR for flooding into the backbone area.

You cannot configure an area to be both a stub area and an NSSA.

The Backbone Area

If you define more than one area in your OSPF network, one of the areas must be designated as the backbone. The backbone area must be of type “normal”, and it must be area 0 (that is, it must have an area ID of 0.0.0.0).

The backbone area must be contiguous, that is, each area in the OSPF autonomous system must have a direct physical connection to the backbone.

An Area Border Router (ABR) must have at least one interface in the backbone area.

Virtual Links

OSPF requires a single contiguous backbone area (area 0.0.0.0). All areas must connect to the backbone, and all inter-area traffic passes through it. If you cannot design your network to have a single contiguous backbone, or if a failed link splits the backbone, you can “repair” or extend a backbone area by creating a virtual link between two non-contiguous areas.

Wherever possible, configure a contiguous backbone and avoid virtual links, because they add complexity to the network. Also, keep the following in mind when creating virtual links:

- A virtual link can traverse one area.
- Virtual links can be created only between Area Border Routers (ABRs).
- Virtual links cannot traverse stub areas or not-so-stubby areas (NSSAs).

OSPF Costs

The OSPF cost is the link-state metric that you want advertised in the link-state advertisement (LSA) as the cost of sending packets over this interface. The cost of reaching any destination is the sum of the costs of the individual hops. You can only assign one cost per interface. The Vyatta system assigns every interface a default cost of 1, which can be modified through configuration.

OSPF Priority

On broadcast interfaces, OSPF uses a router priority value. This priority value is used to determine which routers are selected as the area's Designated Router (DR) and Backup Designated Router (BDR).

The DR and BDR are used to reduce the amount of OSPF traffic on broadcast networks, by reducing the number of adjacent routers to which a router must flood its topology information. In broadcast networks (such as Ethernet), each router establishes an adjacency with only the DR and the BDR, rather than with every router in its area. The DR and the BDR then flood updates from these routers to to all other routers on the network segment.

Priority can range from 0 to 255. In general, the router with the highest priority is elected as the DR, and the router with the second-highest priority is elected as the BDR. The higher the number, the higher the priority. If all routers have an equal priority, the first OSPF router activated on the network becomes the DR and the second router activated becomes the BDR.

Routers with a priority of 0 are ineligible for election.

Route Summaries (Area Ranges)

Area ranges consolidate and summarize internal routes in an area into a single route, for advertisement into an OSPF area border. These are used for route advertisement as type 3 LSAs into other areas of the OSPF domain.

Inter-area route summarization takes place at Area Border Routers (ABRs), and are advertised by ABRs into the backbone area. Therefore, area ranges are configured only on ABRs.

Route summarization helps keep the size of the link state topology database to a minimum, which conserves router memory, minimizes the processing overhead of SPF calculations, reduces the use of network bandwidth by the protocol, and helps speed network convergence.

To summarize the routes in an area into a range, the IP addresses in the area must be assigned so that they are contiguous. This requires careful design of the hierarchical network structure. If there are breaks in the contiguity of IP addresses of an area, you must define more than one route summary for the ranges.

Monitoring the Network

OSPF dynamically discovers the state of the network using LSA updates. It learns about neighbor relationships using hello packets and a router dead interval. The transmit delay introduced by the physical interface itself is also taken into account.

Hello Packets

A hello packet is an OSPF packet used to detect and maintain relationships with neighbors on the same network (directly connected routers). The greater the interval between hello packets, the less router traffic occurs, but the longer it takes for certain topology changes to be detected.

The hello interval must be the same for all routers that are to establish two-way communication within a network. If two routers do not agree on these parameters, they will not consider themselves “neighbors,” and will disregard one another’s communications.

The default hello interval is 10 seconds.

Router Dead Interval

The router dead interval is the maximum time that a router should wait to receiving a hello packet from its neighbor. This value is typically four times the hello interval. If the dead interval passes without the interface receiving a hello packet from the neighbor, the neighbor’s status is changed to out-of-service.

The dead interval must be the same for all routers that are to establish two-way communication within a network. If two routers do not agree on these parameters, they will not consider themselves “neighbors,” and will disregard one another’s communications.

The default router dead interval is 40 seconds.

Interface Transit Delay

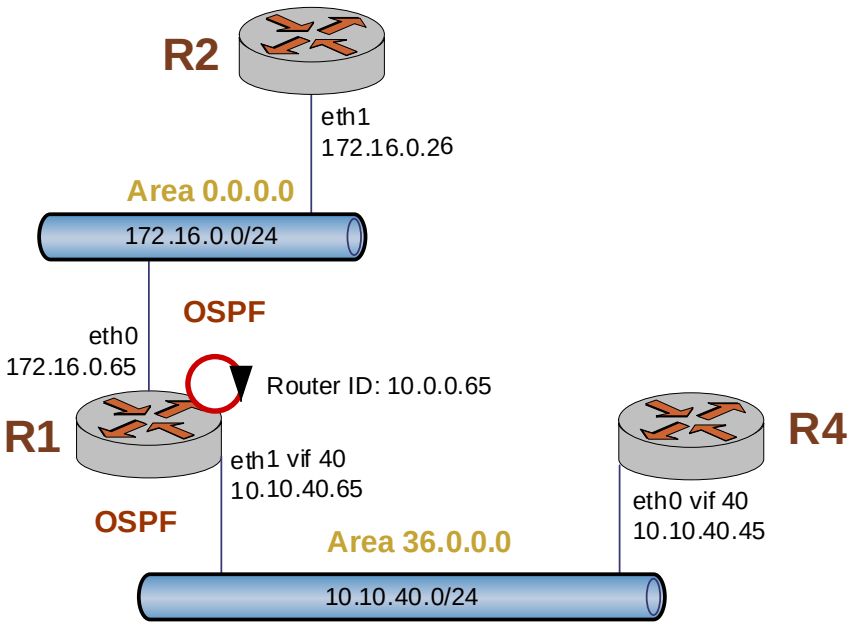
The interface transit delay specifies the estimated time in seconds required to send a link-state advertisement on this interface. The transmit delay is added to the age of the LSA. The LSA age is used to help the network sequence LSAs, so that it can determine which of competing LSAs is the more recent and trustworthy.

LSAs are numbered in sequence, but the sequence numbers are finite, and so cannot be used as the sole determinant of the most recent LSA. Therefore, OSPF also tracks the age of LSAs. Each time the LSA is forwarded to another router, its current age is incremented by the transmit delay. The packet’s age, together with its sequence number, helps the receiving router to determine which version of a received LSA is more recent, and therefore to be used.

Configuring OSPF

In this section, sample configurations are presented for OSPF. When you have finished, the router will be configured as shown in Figure 9-1.

Figure 9-1 OSPF



This section includes the following examples:

- Example 9-1 Basic OSPF Configuration

Basic OSPF Configuration

Example 9-1 establishes one interface in the backbone area and one interface in area 36.0.0.0.

- The router ID is set to the address of the loopback interface. This is considered good practice, as the loopback interface is the most reliable on the router. In this scenario, the loopback address is 10.0.0.65.
- The hello interval for both interfaces is left at the default (10 seconds), as is the dead interval (40 seconds), and the priority (128).

Note the notation for referring to vif 40 of interface eth1: **eth1.40**.

To create a basic OSPF configuration, perform the following steps in configuration mode:

Example 9-1 Basic OSPF Configuration

Step	Command
Create the OSPF configuration node, giving the router ID. This enables OSPF on the router.	`vyatta@R1# set protocols ospf4 router-id 10.0.0.65` `[edit]`
Place 172.16.0.65 on eth0 in the backbone area.	`vyatta@R1# set protocols ospf4 area 0.0.0.0 interface eth0 address 172.16.0.65` `[edit]`
Place 10.10.40.65 on eth1 vif 40 in area 36.0.0.0	`vyatta@R1# set protocols ospf4 area 36.0.0.0 interface eth1.40 address 10.10.40.65` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Redistributing Static and Connected Routes into OSPF

Directly connected routes must be explicitly redistributed by applying a routing policy using the **export** directive within OSPF configuration. You can optionally also redistribute static routes.

“Chapter 6: Routing Policies” provides examples of routing policies applied to various routing protocols, including policies that redistribute static and connected routes. Please see that chapter for more information.

Monitoring OSPF

This section includes the following examples:

- Example 9-2 Showing OSPF routes
- Example 9-3 Showing OSPF neighbor information
- Example 9-4 Showing the OSPF LSA database
- Example 9-5 Viewing the “protocols ospf4” configuration node
- Example 9-6 Enabling logging for OSPF

Showing OSPF Routes

To display route information, use the **show route** command, as in Example 9-2.

Example 9-2 Showing OSPF routes

```
vyatta@R1> show route
Total routes: 6, Total paths: 6
0.0.0.0/0       [static(1)]     > to 10.1.0.1           via eth0
10.1.0.0/24     [connected(0)]  > to 10.1.0.50          via eth0
10.10.10.49/32  [ospf(2)]       > to 10.1.0.49          via eth0
24.0.0.0/8      [ospf(1)]       > to 10.1.0.49          via eth0
172.16.0.0/24   [connected(0)]  > to 172.16.0.50        via eth1
192.168.2.0/24  [ospf(2)]       > to 10.1.0.49          via eth0
```

Showing OSPF Neighbors

To view information about OSPF neighbors, use the **show ospf4 neighbor** command in operational mode.

Example 9-3 Showing OSPF neighbor information

```
vyatta@R1> show ospf4 neighbor
Address      Interface State     ID           Pri  Dead
10.10.30.46  eth0      Full      192.168.2.44 3    39
10.1.0.49    eth0      Full      10.10.10.49  1    37
172.16.0.26  eth0      TwoWay    172.16.0.26  128  36
```

Showing the OSPF LSA Database

To view the OSPF Link State Advertisement (LSA) database, use the **show ospf4 database** command in operational mode.

Example 9-4 Showing the OSPF LSA database

```
vyatta@R1> show ospf4 database
OSPF link state database, Area 0.0.0.0
Type      ID           Adv Rtr      Seq        Age Opt  Cksum  Len
Router    *172.16.0.65 172.16.0.65  0x80000002352 0x2  0x6b5836
Network   10.1.0.49    10.10.10.49  0x800002d0359 0x22 0x923440
Router    10.10.30.46  10.10.30.46  0x800002d0510 0x22 0x518f48
Router    172.16.0.26  172.16.0.26  0x80000005485 0x2  0x5e6348
ASExt-2   24.0.0.0     10.1.0.2     0x800001e2839 0x2  0x66d636
```

Showing OSPF Configuration

You can always view the information in configuration nodes by using the **show** command in configuration mode. In this case, you can view OSPF configuration by using the **show protocols ospf4** command, as shown in Example 9-5.

To show the information in the **protocols ospf4** configuration node, perform the following step in configuration mode:

Example 9-5 Viewing the “protocols ospf4” configuration node

Step	Command

Show the contents of the **protocols ospf4** configuration node.	`vyatta@R1# show protocols ospf4`
	`router-id: 127.0.0.1`
	`area 0.0.0.0 {`
	`interface eth0 {`
	`address 172.16.0.65 {`
	`}`
	`}`
	`interface eth1 {`
	`address 10.10.30.65 {`
	`}`
	`}`
	`}`

Sending OSPF Messages to Syslog

Creating and enabling the **protocols ospf4 traceoptions** configuration node directs the OSPF process to produce OSPF-specific log messages. When OSPF logging is enabled, the log messages are sent to whatever destinations are configured for syslog.

By default, log messages are sent to the main log file at **/var/log/messages**. However, you can configure syslog to send messages to other destinations, such as a user-specified file, the console, a remote host, or a user account.

For more information about logging, please see “Chapter 16: Logging.”

Example 9-6 enables logging for OSPF.

To enable logging for OSPF, perform the following steps in configuration mode:

Example 9-6 Enabling logging for OSPF

Step	Command

Enable logging within the **ospf** configuration node.	`vyatta@R1# set protocols ospf traceoptions flag all disable false`
	`OK`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Chapter 10: BGP

This chapter describes how to configure the Border Gateway Protocol on the Vyatta system.

The following topics are covered:

- BGP Overview
- Supported Standards
- Configuring BGP
- Monitoring BGP

BGP Overview

This section presents the following topics:

- iBGP and eBGP
- Scalability of BGP
- Confederations
- Route Reflection
- Route Flapping and Flap Damping
- AS Paths
- BGP Communities

Border Gateway Protocol (BGP) is the principal inter-domain routing protocol used on the Internet. BGP version 4 is specified in RFC 4271, which obsoletes the original BGPv4 specification defined in RFC 1771.

The principal concept of BGP is that of the Autonomous System (AS). An AS is a routing domain that is under one administrative authority, and which implements its own routing policies. For example, one Internet Service Provider (ISP) would have its own AS, while another would have its own, different, AS. Many large enterprises also have their own AS, particularly if they are multi-homed (that is, connected to multiple ISPs). The BGP routing protocol is used to convey network reachability information between ASs.

Routers that are configured to run BGP between one another are known as BGP peers or BGP neighbors. BGP uses a TCP connection on the well-known port 179 to exchange routing information between peers. BGP peers that are configured within the same AS are referred to as internal BGP (iBGP) peers. BGP peers that are configured in different ASs are referred to as external BGP (eBGP) peers.

There are two basic types of BGP route exchanges that occur between peers: route announcements and route withdrawals.

- A route announcement tells a peer that it can reach a particular network via the announcing router, and includes attributes associated with that path.
- A route withdrawal tells a peer that a previously announced route is no longer reachable via this peer.

All valid route announcements that are received on a BGP router are placed into the router's BGP table. (These routes are typically referred to as BGP paths.) This means that, for a particular network prefix—for example, 10.0.0.0/8—the local BGP router might have recorded multiple available paths: one through any of its BGP peers. For each prefix, the BGP process uses a path selection algorithm to select the best available path from all those learned from its peers. Once the best path has been selected, that path becomes the candidate route from the BGP protocol for inserting into the active routing table.

Each BGP path includes several attributes that are used by the BGP path selection process to determine which path is the best available path. These attributes can also be used in user-defined routing policies applied to BGP; these can allow the router to perform additional actions on a matching path, such as determining whether to accept or reject a route announcement.

One of the most commonly used BGP path attributes is the AS path. The AS path lists each of the ASs by which the prefix has been announced, and is used to prevent routing loops. The AS path is read from right to left, where the right-most AS is the AS that originated the network prefix (that is, it was the first AS to announce reachability for this prefix). This AS is known as the origin AS.

As a network prefix is advertised between ASs, each AS prepends its own AS number to the AS path. For example, the AS path “4 3 2 1” indicates that AS 1 originated the network prefix. The prefix was advertised from AS 1 to AS 2, then from AS 2 to AS 3, and finally from AS 3 to AS 4.

Other BGP path attributes include origin, next hop, multi-exit discriminator (“med”), local preference (“local pref”), atomic aggregate, and aggregator. These attributes are described in more detail in another section of this document.

iBGP and eBGP

A BGP peer can be one of two types:

- Internal BGP (iBGP) peers are peers that are configured with the same AS number.
- External BGP (eBGP) peers are peers that are configured with different AS numbers.

iBGP

The BGP protocol requires that all iBGP peers within an AS have a connection to one another, creating a full-mesh of iBGP peering connections. (The exception to this is route reflection.) When a prefix is announced from one iBGP peer to another, the AS path is not changed. Due to the full-mesh requirement, all iBGP peers should have the same view of the BGP table, unless different routing policies have been applied to some of the peers.

When a router receives an iBGP announcement, the BGP process uses the BGP best path selection algorithm to determine whether the received announcement is the best available path for that prefix. If it is the best available path, then the BGP process uses this route as the BGP candidate route for insertion into the routing table, and the BGP process announces this path to all its peers, both iBGP and eBGP peers. If it is not the best available path, then the BGP process keeps a copy of this path in its BGP table, so that it can be used to calculate the best available path when path information for that prefix changes (for example, if the current best available path is withdrawn).

The BGP ID is a unique identifier in the format of an IP address used to identify a peer. The peering IP address is the actual IP address used for the BGP connection.

For iBGP peerings, the BGP ID and peering IP is frequently the IP address bound to that router's loopback interface. An iBGP session is usually contained within a local LAN, with multiple redundant physical links between the iBGP devices. For iBGP routes, reachability is all that is necessary, and the loopback interface is reachable so long as at least one physical interface is operational. Because of the physical and/or logical redundancy that exists between iBGP peers, iBGP peering on the loopback interface works well.

Since BGP does not provide reachability information, you must make sure that each iBGP peer knows how to reach other peers. To be able to reach one another, each peer must have some sort of Interior Gateway Protocol (IGP) route, such as a connected route, a static route, or a route through a dynamic routing protocol such as RIP or OSPF, which tells them how to reach the opposite router.

eBGP

External BGP is the method that different Autonomous Systems (ASs) use to interconnect with one another. eBGP usually takes place over WAN links, where there may be a single physical path between eBGP peers. Alternatively, they may have multiple eBGP peer connections to provide redundancy and/or traffic load balancing. Redundant peers use distinct BGP sessions so that, if one session fails, another can take over.

BGP uses an AS path to track the path of a prefix through the various ASs that send or receive the prefix announcement. When a prefix is announced to an eBGP peer, the local AS number is prepended to the AS path. This helps to prevent routing loops by rejecting any prefix announcements that include the local AS number in the AS path. Prefix announcements learned via eBGP are also analyzed using the BGP best path selection process.

For eBGP peerings, the BGP ID and peering IP address is typically the local IP address of the interface that is being used to connect to the eBGP peers. However if more than one physical interface is being used for eBGP peering it is also common to use a loopback IP address as the BGP ID, but still use the physical interface IP address as the peering IP address.

BGP Path Selection Process

The BGP process may receive advertisements from multiple peers for the same network prefix. Each of these announcements from a peer for a prefix is called a path. The BGP process selects the “best” path from all available paths and this path becomes the candidate route announced by BGP for inclusion in the Routing Information Base (RIB).

Depending on what other protocols also have candidate routes for this network prefix, the BGP route may or may not be added to the RIB. For instance if the RIB has candidate routes from both BGP and static routing for the same prefix, the static route, not the BGP route, will be included in the RIB. This is because the static route process has a lower administrative cost than the BGP process.

It is important to note that BGP will not consider paths that contain a NEXT_HOP value that is not reachable via an entry in the RIB. For all valid paths, the Vyatta router uses a BGP path selection process based on decision process described in RFC 4271, section 9.1. BGP paths are preferred based on the following:

- **LOCAL PREFERENCE**: Prefer the path with the highest LOCAL_PREF
- **AS PATH LENGTH**: Prefer the path with the shortest AS_PATH.
- **ORIGIN**: Prefer the path with the lowest ORIGIN type1.
- **MULTI_EXIT_DISC**: Prefer the path with the lowest MED2.
- **PEER TYPE**: Prefer paths learned via eBGP over paths learned via iBGP.
- **IGP METRIC**. Prefer paths with lower IGP metric for the path’s NEXT_HOP address.
- **BGP ID**: Prefer the path with the lowest BGP ID.
- **PEER IP**: Prefer the path with the lowest peer IP address.

The best path selection process is performed as “first match and out.” This means that two paths will be compared until the first difference in preference criteria. For example, two paths for the same network prefix may have the same LOCAL_PREF value, but different AS path lengths. In this case, the path with the shortest AS path would be the “best” path. If the peer IP address is being used to select the best path, this means that all other path criteria were the same for the available paths.

You can use the **show bgp routes** command to see the current best paths in the RIB.

Scalability of BGP

The Border Gateway Protocol 4 specification (RFC 4271) requires that iBGP peers be fully meshed; that is, every iBGP peer must have a connection to every other iBGP peer. A full mesh of iBGP peers does not scale well to large ASs, which can have hundreds of iBGP routers. To overcome scalability issues, two enhancements have been developed for BGP:

- BGP Confederations (RFC 3065)
- Route Reflection (RFC 2796)

Confederations

Confederations enable you to reduce the size and complexity of the iBGP mesh. In a BGP confederation, a single AS is divided into multiple internal sub-ASs to help keep the number of iBGP peer connections manageable. Each sub-AS is assigned its own AS number; this is typically assigned from the private AS number space, which ranges from 65412 to 65535. Within a sub-AS, all the standard iBGP rules, including full-mesh peering, apply. The connections between confederation sub-ASs use eBGP peering. One or more eBGP connections can be made between each sub-AS. The sub-ASs are grouped as a confederation, which advertises as a single AS to external peers.

Figure 10-1 shows the large number of iBGP connections that must be configured in even a moderately sized AS. In this example, 14 routers are participating in iBGP.

Figure 10-1 iBGP full mesh

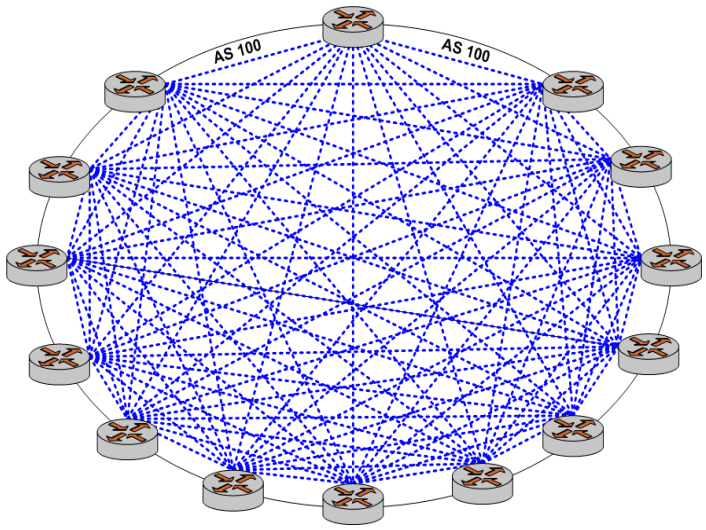
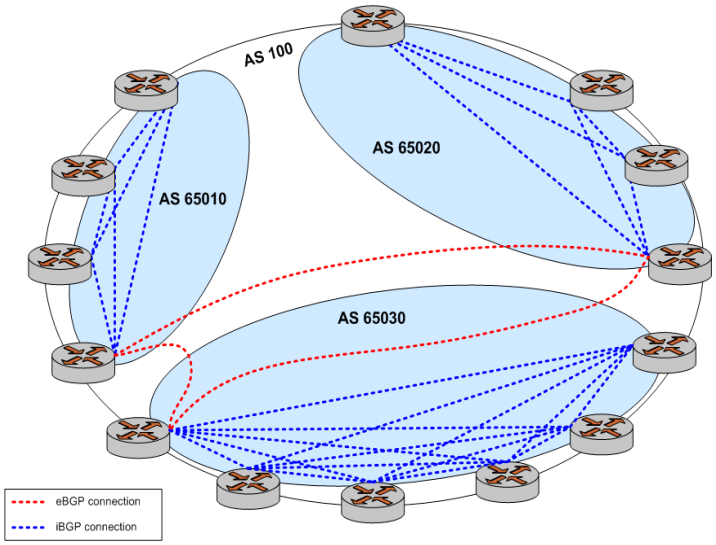


Figure 10-2 shows a BGP confederation that splits the single AS shown in Figure 10-1 into three sub-ASs, which each use private AS numbers. Within each sub-AS, all of the iBGP peers are fully meshed. The sub-ASs are connected to one another using an eBGP connection.

Figure 10-2 BGP confederation



Route Reflection

Another technology designed to help ASs with large numbers of iBGP peers is route reflection. In a standard BGP implementation, all iBGP peers must be fully meshed. because of this requirement, when an iBGP peer learns a route from another iBGP peer, the receiving router does not forward the route to any of its iBGP peers, since these routers should have learned the route directly from the announcing router.

In a route reflector environment the iBGP peers are no longer fully meshed. Instead, each iBGP peer has an iBGP connection to one or more route reflector (RR) servers. Routers configured with a connection to an RR server are referred to as RR clients. Only the RR server is configured to be aware that the RR client is part of an RR configuration; from the RR client's point of view, it is configured normally, and does not have any awareness that it is part of a RR configuration.

In route reflection, internal peers of an RR server are categorized into two types:

- **Client peers.** The RR server and its client peers form a cluster. Within a cluster, client peers need not be fully meshed, but must have an iBGP connection to at least one RR in the cluster.
- **Non-client peers.** Non-client peers, including the RR server, must be fully meshed.

An RR environment is unlike a regular environment, where iBGP peers never forward a route update to other iBGP peers (which is the reason why each iBGP peer must peer with all other peers). When an RR server receives an iBGP update from an RR client, these route updates can also be sent to all other RR clients. When an RR server receives a route update from a peer, it selects the best path based on its path selection rule. After the best path is selected, the RR server chooses its action depending on the type of the peer from which it learned the best path.

- If the route was learned from a client peer, the RR reflects the route to both client and non-client peers. All iBGP updates from client peers are reflected to all other client peers in the cluster. This is done regardless of whether the update was the best path for the RR itself.
- If the route was learned from a non-client iBGP peer, it is reflected out to all RR client peers.
- If the route was learned from an eBGP peer, the route is reflected to all RR clients and all non-clients.

Figure 10-3 shows again the full mesh of iBGP connections in even a moderately sized AS.

Figure 10-3 iBGP full mesh

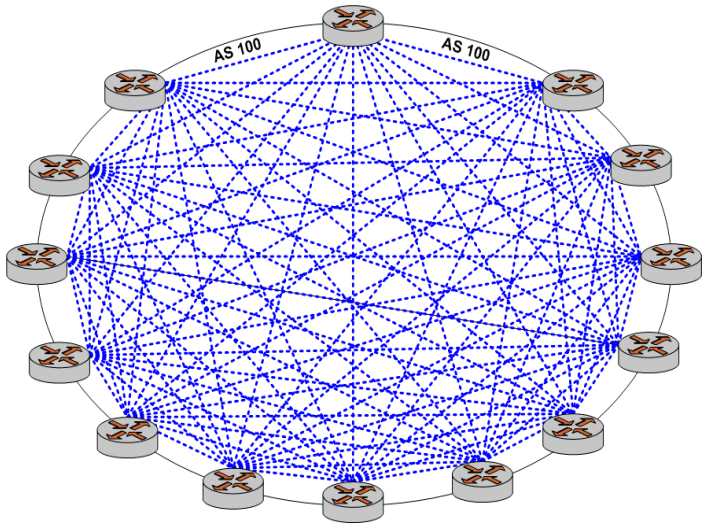
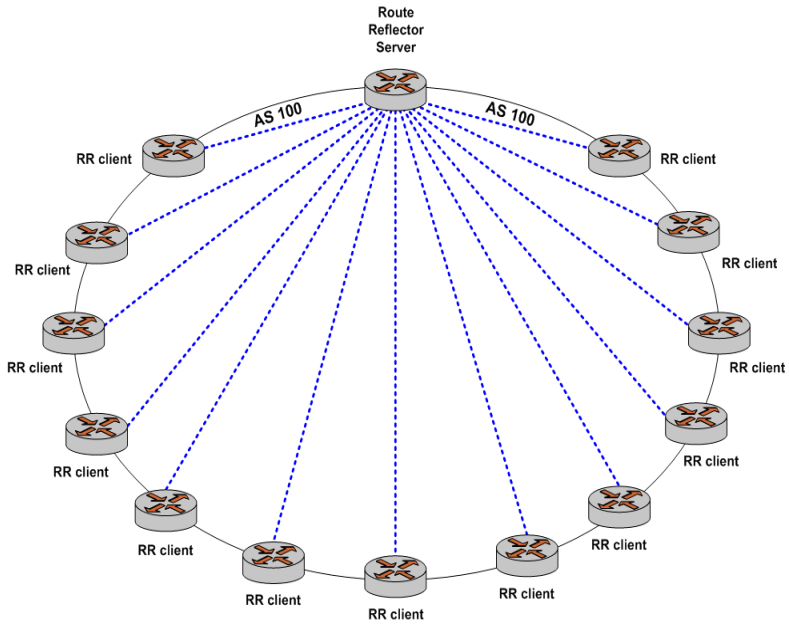


Figure 10-4 shows how introducing route reflection into the AS dramatically reduces the number of iBGP connections required within the AS.

Figure 10-4 iBGP route reflection



Note that to prevent looping, clients must not peer with RRs outside of the cluster.

To achieve redundancy, more than one RR server can be configured within a cluster. Also, to scale to very large networks, a large AS can be configured to have multiple clusters with redundant RR servers, where the RR servers are all configured with a full mesh of iBGP connections between the RR servers.

Route Flapping and Flap Damping

Route flapping is a situation where a route fluctuates repeatedly between being announced, then withdrawn, then announced, then withdrawn, and so on. In this situation, a BGP system will send an excessive number of update messages advertising network reachability information.

Route flapping can cause several different issues. First, each time a new route is learned or withdrawn the BGP best path selection process for that prefix must be executed, which can result in high CPU utilization. If a large enough number of routes are flapping, the BGP process may not be able to converge sufficiently quickly. Second, the route flapping issue

can become amplified as it passes from peer to peer. For example, if a router with two peers flaps a route, and those two peers each have 10 peers, the flapping route affects 20 BGP routers.

Route dampening is intended to minimize the propagation of update messages between BGP peers for flapping routes. This reduces the load on these devices without unduly impacting the route convergence time for stable routes.

When route damping is enabled, a route is assigned a penalty each time it "flaps" (that is, each time it is announced and then withdrawn within a short interval). If the penalty exceeds a configured threshold (its *suppress* value) the route is suppressed.

After the route has been stable for a configured interval (its *half-life*) the penalty is reduced by half. Subsequently, the penalty is reduced every five seconds. When the penalty falls below a configured value (its *reuse* value), the route is unsuppressed.

The penalty applied to a route will never exceed the *maximum penalty*, which is computed from configured attributes as follows:

```
Maximum penalty = reuse * 2^(suppress/half-life)
```

While the route is being "damped," updates and withdrawals for this route from a peer are ignored. This helps to localize the route flapping to a particular peering connection.

AS Paths

An AS path is a path to a destination in the Border Gateway Protocol (BGP). The path is represented as a sequence of AS numbers, which are the numbers uniquely identifying BGP autonomous systems. Each AS number represents an autonomous system (which may be comprised of multiple networks) that a packet traverses if it takes the associated route to the destination.

For a packet to reach a destination using this route, it traverses the listed ASs from the leftmost AS number to the rightmost, where the rightmost is the AS immediately preceding its destination.

Using policies, match conditions can be defined based on all or portions of the AS path. To do this, you can either specify the AS path directly in a policy command using a regular expression in the **as-path** attribute, or create a named AS path regular expression using the **as-path-list** attribute and including the name in a policy command.

For information on specifying AS paths using regular expressions, please see the section "Defining a BGP AS Path List" in "Chapter 6: Routing Policies."

BGP Communities

All BGP updates include a BGP attribute called the communities path attribute. The communities path attribute allows ASs to “tag” prefix announcements. This tag can then be used by routing policies to modify the normal behavior for that prefix announcement. For example, an AS could choose to filter out all prefix announcements containing a community value that identifies the prefix as a customer-generated prefix, instead only announcing the summary prefix for all customer prefixes. It is important to note that the community path attribute is carried in BGP update messages, which allows ASs not directly connected to each other to share information about a prefix.

The format for community identifiers is defined in RFC 1997: “BGP Communities Attribute.” The community identifier is a 32-bit value, where the first two bytes of the value are the AS number and the second two bytes are an arbitrary value defined by the AS. This format can be represented as *AA:NN*, where *AA* is the AS number of the AS adding the community identifier to the community path attribute, and *NN* represents a user-defined policy value.

There are two types of BGP communities: “well-known” communities and user-defined or private communities.

The Vyatta router recognizes the following BGP well-known communities as per RFC 1997:

NO_EXPORT: All routes received carrying a communities attribute containing this value are not advertised outside a BGP confederation boundary (a stand-alone autonomous system that is not part of a confederation should be considered a confederation itself).

NO_ADVERTISE: All routes received carrying a communities attribute containing this value are not advertised to other BGP peers.

NO_EXPORT_SUBCONFED: All routes received carrying a communities attribute containing this value are not advertised to external BGP peers (this includes peers in other members autonomous systems inside a BGP confederation).

Supported Standards

The Vyatta implementation of BGP complies with the following standards:

- RFC 4271: BGP-4 Specification
- RFC 4273: Definitions of Managed Objects for BGP-4
- RFC 1997: BGP Communities Attribute
- RFC 3065: BGP Confederations RFC 3065
- RFC 2796: Route Reflection RFC 2796

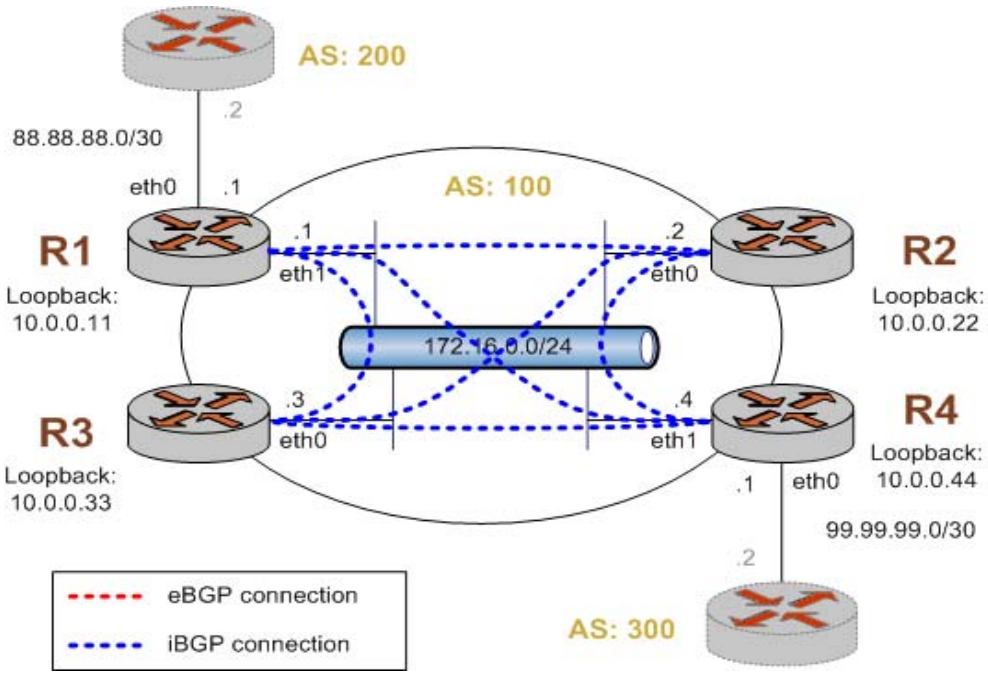
Configuring BGP

This section presents the following topics:

- Basic iBGP Configuration
- Verifying the iBGP Configuration
- Basic eBGP Configuration
- Verifying the eBGP Configuration
- Originating a Route to eBGP Neighbors
- Verifying the Route Origination
- Inbound Route Filtering
- Verifying the Inbound Filter
- Outbound Route Filtering
- Verifying the Outbound Filter
- Confederations
- Verifying the Confederation
- Route Reflectors
- Verifying the Route Reflector
- Route Redirection

This section presents sample configurations for BGP. The configuration examples are based on the reference diagram in Figure 10-5.

Figure 10-5 BGP configuration reference diagram



Basic iBGP Configuration

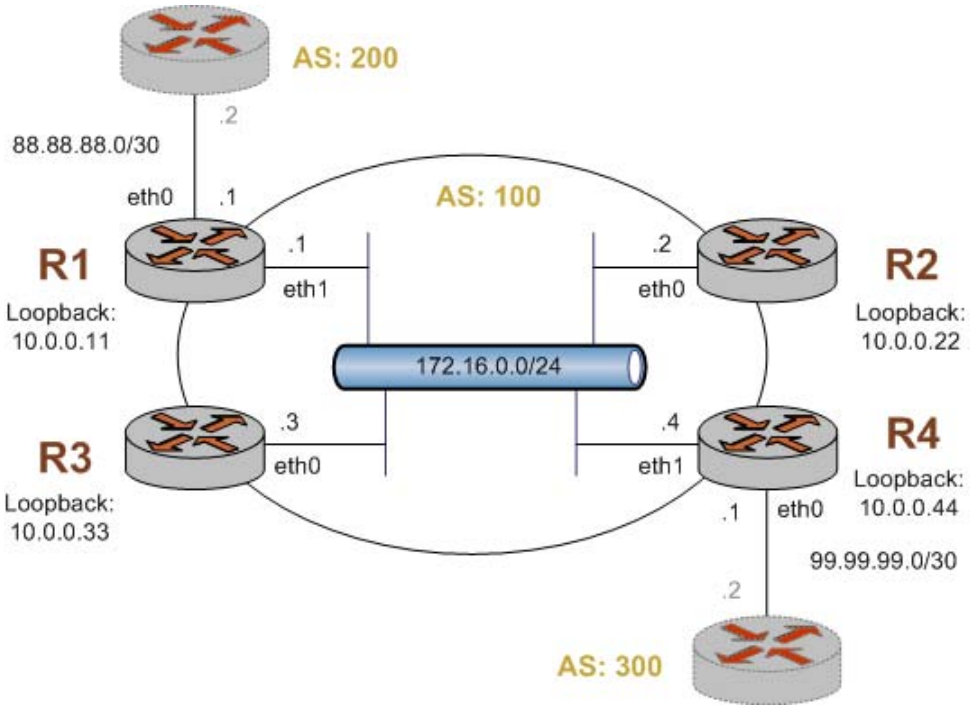
In this section, you configure iBGP on the routers labeled R1, R2, R3, and R4 in the reference network diagram. Each router has an iBGP peering connection to each of the other iBGP routers in the network, satisfying the full mesh iBGP peering requirement.

In the example the iBGP peering connections are established between iBGP neighbors using the loopback interface IP addresses. This is typical practice, particularly when there are redundant connections between the iBGP routers.

In order for the routers to be able to reach each other using the loopback IP address, the loopback IP addresses must be reachable via an entry in the router's routing table. This requires some form of Internal Gateway Protocol (IGP). In the example here, we will use a basic Open Shortest Path First (OSPF) configuration to announce the loopback addresses between neighbors.

Figure 10-6 shows the BGP connections after you have completed the iBGP configuration.

Figure 10-6 Basic iBGP configuration



This example assumes that you have already configured the router interfaces; only the steps required to implement BGP are shown.

To create a basic iBGP configuration, perform the following steps in configuration mode:

Example 10-1 Basic iBGP configuration

Router	Step	Command(s)
R1	Enable OSPF and set the router ID to be the loopback IP address.	`vyatta@R1# set protocols ospf4 router-id 10.0.0.11` `[edit]`
R1	Configure the loopback interface to be in OSPF area 0.0.0.0.	`vyatta@R1# set protocols ospf4 area 0.0.0.0 interface lo address 10.0.0.11` `[edit]`
R1	Configure eth0 interface to be in OSPF area 0.0.0.0.	`vyatta@R1# set protocols ospf4 area 0.0.0.0 interface eth1 address 172.16.0.1` `[edit]`
R1	Configure the eth1 interface to be in OSPF area 0.0.0.0 and make it a passive interface.	`vyatta@R1# set protocols ospf4 area 0.0.0.0 interface eth0 address 88.88.88.1 passive true` `[edit]`

Example 10-1 Basic iBGP configuration

R1	Create the BGP configuration node. This enables BGP on the router. Set the BGP ID to the loopback address, which on R1 is 10.0.0.11.	`vyatta@R1# set protocols bgp bgp-id 10.0.0.11` `[edit]`
R1	Specify the local AS for router R1.	`vyatta@R1# set protocols bgp local-as 100` `[edit]`
R1	Create an iBGP peer for R2. The peer is an iBGP peer because it resides within the same AS as this router.	`vyatta@R1# set protocols bgp peer 10.0.0.22 as 100` `[edit]`
R1	Define the IP address on the local R1 router that is used to peer with the R2 router.	`vyatta@R1# set protocols bgp peer 10.0.0.22 local-ip 10.0.0.11` `[edit]`
R1	Defines the next hop that will be set for route updates sent to peer R2.	`vyatta@R1# set protocols bgp peer 10.0.0.22 next-hop 10.0.0.11` `[edit]`
R1	Create an iBGP peer for R3. The peer is an iBGP peer because it resides within the same AS as this router.	`vyatta@R1# set protocols bgp peer 10.0.0.33 as 100` `[edit]`
R1	Define the IP address on the local R1 router that is used to peer with the R3 router.	`vyatta@R1# set protocols bgp peer 10.0.0.33 local-ip 10.0.0.11` `[edit]`
R1	Defines the next hop that will be set for route updates sent to peer R3.	`vyatta@R1# set protocols bgp peer 10.0.0.33 next-hop 10.0.0.11` `[edit]`
R1	Create an iBGP peer for R4. The peer is an iBGP peer because it resides within the same AS as this router.	`vyatta@R1# set protocols bgp peer 10.0.0.44 as 100` `[edit]`
R1	Define the IP address on the local R1 router that is used to peer with the R4 router.	`vyatta@R1# set protocols bgp peer 10.0.0.44 local-ip 10.0.0.11` `[edit]`
R1	Defines the next hop that will be set for route updates sent to peer R4.	`vyatta@R1# set protocols bgp peer 10.0.0.44 next-hop 10.0.0.11` `[edit]`

Example 10-1 Basic iBGP configuration

R1	Commit the configuration.	`vyatta@R1# commit`
		`OK`
		`[edit]`

R2	Enable OSPF and set the router id to be the loopback IP address.	`vyatta@R2# set protocols ospf4 router-id 10.0.0.22` `[edit]`
R2	Configure the loopback interface to be in OSPF area 0.0.0.0.	`vyatta@R2# set protocols ospf4 area 0.0.0.0 interface lo address 10.0.0.22` `[edit]`
R2	Configure eth1 interface to be in OSPF area 0.0.0.0.	`vyatta@R2# set protocols ospf4 area 0.0.0.0 interface eth1 address 172.16.0.2` `[edit]`
R2	Create the BGP configuration node. This enables BGP on the router. Set the BGP ID to the loopback address, which on R2 is 10.0.0.22.	`vyatta@R2# set protocols bgp bgp-id 10.0.0.22` `[edit]`
R2	Specify the local AS for router R2.	`vyatta@R2# set protocols bgp local-as 100` `[edit]`
R2	Create an iBGP peer for R1. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R2# set protocols bgp peer 10.0.0.11 as 100` `[edit]`
R2	Define the IP address on the local R2 router that is used to peer with the R1 router.	`vyatta@R2# set protocols bgp peer 10.0.0.11 local-ip 10.0.0.22` `[edit]`
R2	Defines the next hop that will be set for route updates sent to peer R1.	`vyatta@R2# set protocols bgp peer 10.0.0.11 next-hop 10.0.0.22` `[edit]`
R2	Create an iBGP peer for R3. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R2# set protocols bgp peer 10.0.0.33 as 100` `[edit]`
R2	Define the IP address on the local R2 router that is used to peer with the R3 router.	`vyatta@R2# set protocols bgp peer 10.0.0.33 local-ip 10.0.0.22` `[edit]`

Example 10-1 Basic iBGP configuration

R2	Defines the next hop that will be set for route updates sent to peer R3.	`vyatta@R2# set protocols bgp peer 10.0.0.33 next-hop 10.0.0.22` `[edit]`
R2	Create an iBGP peer for R4. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R2# set protocols bgp peer 10.0.0.44 as 100` `[edit]`
R2	Define the IP address on the local R2 router that is used to peer with the R4 router.	`vyatta@R2# set protocols bgp peer 10.0.0.44 local-ip 10.0.0.22` `[edit]`
R2	Defines the next hop that will be set for route updates sent to peer R4.	`vyatta@R2# set protocols bgp peer 10.0.0.44 next-hop 10.0.0.22` `[edit]`

R2	Commit the configuration.	`vyatta@R2# commit`
		`OK`
		`[edit]`

R3	Enable OSPF and set the router id to be the loopback IP address.	`vyatta@R3# set protocols ospf4 router-id 10.0.0.33` `[edit]`
R3	Configure the loopback interface to be in OSPF area 0.0.0.0	`vyatta@R3# set protocols ospf4 area 0.0.0.0 interface lo address 10.0.0.33` `[edit]`
R3	Configure eth0 interface to be in OSPF area 0.0.0.0.	`vyatta@R3# set protocols ospf4 area 0.0.0.0 interface eth0 address 172.16.0.3` `[edit]`
R3	Create the BGP configuration node. This enables BGP on the router. Set the BGP ID to the loopback address, which on R3 is 10.0.0.33.	`vyatta@R3# set protocols bgp bgp-id 10.0.0.33` `[edit]`
R3	Specify the local AS for router R3.	`vyatta@R3# set protocols bgp local-as 100` `[edit]`
R3	Create an iBGP peer for R1. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R3# set protocols bgp peer 10.0.0.11 as 100` `[edit]`

Example 10-1 Basic iBGP configuration

R3	Define the IP address on the local R3 router that is used to peer with the R1 router.	`vyatta@R3# set protocols bgp peer 10.0.0.11 local-ip 10.0.0.33` `[edit]`
R3	Defines the next hop that will be set for route updates sent to peer R1.	`vyatta@R3# set protocols bgp peer 10.0.0.11 next-hop 10.0.0.33` `[edit]`
R3	Create an iBGP peer for R2. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R3# set protocols bgp peer 10.0.0.22 as 100` `[edit]`
R3	Define the IP address on the local R3 router that is used to peer with the R2 router.	`vyatta@R3# set protocols bgp peer 10.0.0.22 local-ip 10.0.0.33` `[edit]`
R3	Defines the next hop that will be set for route updates sent to peer R2.	`vyatta@R3# set protocols bgp peer 10.0.0.22 next-hop 10.0.0.33` `[edit]`
R3	Create an iBGP peer for R4. The peer is an iBGP peer because it resides within the same AS as the router.	`vyatta@R3# set protocols bgp peer 10.0.0.44 as 100` `[edit]`
R3	Define the IP address on the local R3 router that is used to peer with the R4 router.	`vyatta@R3# set protocols bgp peer 10.0.0.44 local-ip 10.0.0.33` `[edit]`
R3	Defines the next hop that will be set for route updates sent to peer R4.	`vyatta@R3# set protocols bgp peer 10.0.0.44 next-hop 10.0.0.33` `[edit]`

R3	Commit the configuration.	`vyatta@R3# commit`
		`OK`
		`[edit]`

R4	Enable OSPF and set the router id to be the loopback IP address.	`vyatta@R4# set protocols ospf4 router-id 10.0.0.44` `[edit]`
R4	Configure the_loopback lo interface to be in OSPF area 0.0.0.0.	`vyatta@R4# set protocols ospf4 area 0.0.0.0 interface lo address 10.0.0.44` `[edit]`

Example 10-1 Basic iBGP configuration

R4	Configure eth1 interface to be in OSPF area 0.0.0.0.	vyatta@R4# **set protocols ospf4 area 0.0.0.0 interface eth1 address 172.16.0.4** [edit]
R4	Configure eth0 interface to be in OSPF area 0.0.0.0 and make it a passive interface.	vyatta@R4# **set protocols ospf4 area 0.0.0.0 interface eth0 address 99.99.99.1 passive true** [edit]
R4	Create the BGP configuration node. This enables BGP on the router. Set the BGP ID to the loopback address, which on R4 is 10.0.0.44.	vyatta@R4# **set protocols bgp bgp-id 10.0.0.44** [edit]
R4	Specify the local AS for router R4.	vyatta@R4# **set protocols bgp local-as 100** [edit]
R4	Create an iBGP peer for R1. The peer is an iBGP peer because it resides within the same AS as the router.	vyatta@R4# **set protocols bgp peer 10.0.0.11 as 100** [edit]
R4	Define the IP address on the local R4 router that is used to peer with the R1 router.	vyatta@R4# **set protocols bgp peer 10.0.0.11 local-ip 10.0.0.44** [edit]
R4	Defines the next hop that will be set for route updates sent to peer R1.	vyatta@R4# **set protocols bgp peer 10.0.0.11 next-hop 10.0.0.44** [edit]
R4	Create an iBGP peer for R2. The peer is an iBGP peer because it resides within the same AS as the router.	vyatta@R4# **set protocols bgp peer 10.0.0.22 as 100** [edit]
R4	Define the IP address on the local R4 router that is used to peer with the R2 router.	vyatta@R4# **set protocols bgp peer 10.0.0.22 local-ip 10.0.0.44** [edit]
R4	Defines the next hop that will be set for route updates sent to peer R2.	vyatta@R4# **set protocols bgp peer 10.0.0.22 next-hop 10.0.0.44** [edit]
R4	Create an iBGP peer for R3. The peer is an iBGP peer because it resides within the same AS as the router.	vyatta@R4# **set protocols bgp peer 10.0.0.33 as 100** [edit]

Example 10-1 Basic iBGP configuration

R4	Define the IP address on the local R4 router that is used to peer with the R3 router.	`vyatta@R4# set protocols bgp peer 10.0.0.33 local-ip 10.0.0.44` `[edit]`
R4	Defines the next hop that will be set for route updates sent to peer R3.	`vyatta@R4# set protocols bgp peer 10.0.0.33 next-hop 10.0.0.44` `[edit]`

R4	Commit the configuration.	`vyatta@R4# commit`
		`OK`
		`[edit]`

Verifying the iBGP Configuration

The following commands can be used to verify the iBGP configuration.

R1: show bgp peers

Example 10-2 shows the output of the **show bgp peers** command for router R1 at this stage of the configuration.

Example 10-2 Verifying iBGP on R1: ”show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.22        100    ESTAB     4    22         38       0         0
10.0.0.33        100    ESTAB     4    26         25       0         0
10.0.0.44        100    ESTAB     4    16         15       0         0

vyatta@R1>
```

The most important field in the output for **show bgp peers** is the **State** field. All the iBGP peers for R1 are in the ESTAB state, which stands for “established.” The established state indicates that the peers have successfully created a BGP connection between one another, and are now able to send and receive BGP update messages.

If a peer shows in either ACTIVE or IDLE states, it means there is some issue that is keeping the BGP peers from forming an adjacency.

- The ACTIVE state identifies that the local router is actively trying to establish a TCP connection to the remote peer. You may see this if the local peer has been configured, but the remote peer is unreachable or has not been configured.
- The IDLE state indicates that the local router has not allocated any resources for that peer connection, so any incoming connection requests will be refused.

R1: show bgp neighbor-routes

Because we have not configured any routing announcements yet, the BGP table is currently empty. This can be seen by the output of **show bgp neighbor-routes** for R1, which is shown in Example 10-3.

Example 10-3 Verifying iBGP on R1: “show bgp neighbor-routes”

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

   Prefix           Nexthop      Peer        AS Path
   ------           -------      ----        -------
*> 172.20.1.1/32    172.21.1.11  172.20.1.1  65511 i
*> 172.21.1.0/24    172.21.1.11  172.20.1.1  65511 i
*> 172.21.111.0/24  172.21.1.11  172.20.1.1  65511 i

vyatta@R1>
```

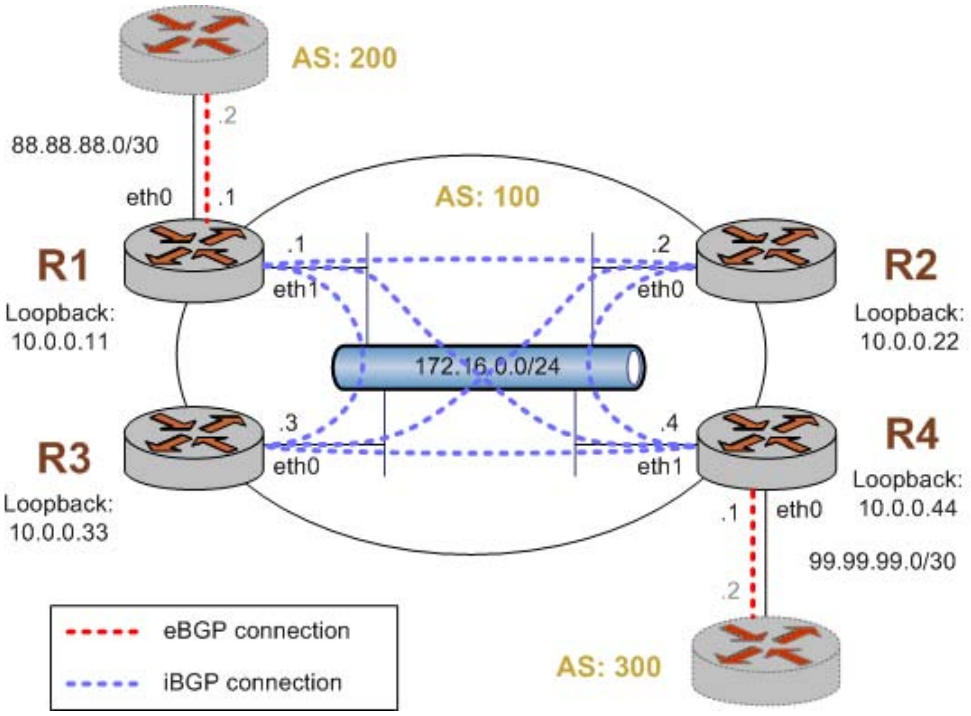
Basic eBGP Configuration

In this section, you configure eBGP on the routers labeled R1 and R4 in the reference network diagram. Router R1 is peering with an eBGP neighbor that is configured to be in AS 200 and router R4 is peering with an eBGP neighbor in AS 300.

In this example, the eBGP peering connections are established between eBGP neighbors using the physical interface IP addresses. This is a common configuration for eBGP peers. If the link between the peers goes down, the peering relationship should also go down since there is no redundancy.

After the basic eBGP configuration has been completed, the network should look as shown in Figure 10-7.

Figure 10-7 Basic eBGP configuration



This example assumes the following:

- The configuration in Example 10-6 has already been performed.
- The eBGP peers connecting to R1 and R4 have been properly configured for BGP.

To create a basic eBGP configuration, perform the following steps in configuration mode:

Example 10-4 Basic eBGP configuration

Router	Step	Command(s)
R1	Create an eBGP peer for the router connected to eth0 on R1.	`vyatta@R1# set protocols bgp peer 88.88.88.2 as 200` `[edit]`
R1	Define the IP address on the local R1 router that is used to peer with the eBGP neighbor.	`vyatta@R1# set protocols bgp peer 88.88.88.2 local-ip 88.88.88.1` `[edit]`
R1	Defines the next hop that will be set for route updates sent to the eBGP peer.	`vyatta@R1# set protocols bgp peer 88.88.88.2 next-hop 88.88.88.1` `[edit]`

Example 10-4 Basic eBGP configuration

R1	Commit the configuration.	`vyatta@R1# commit`
		`OK`
		`[edit]`

R4	Create an eBGP peer for the router connected to eth0 on R4.	`vyatta@R4# set protocols bgp peer 99.99.99.2 as 300` `[edit]`
R4	Define the IP address on the local R4 router that is used to peer with the eBGP neighbor.	`vyatta@R4# set protocols bgp peer 99.99.99.2 local-ip 99.99.99.1` `[edit]`
R4	Defines the next hop that will be set for route updates sent to the eBGP peer.	`vyatta@R4# set protocols bgp peer 99.99.99.2 next-hop 99.99.99.1` `[edit]`

R4	Commit the configuration.	`vyatta@R4# commit`
		`OK`
		`[edit]`

Verifying the eBGP Configuration

The following commands can be used to verify the eBGP configuration. Note that the output shown for these commands would be obtained *after* the configuration for both router R1 and router R4 has been completed.

R1: show bgp peers

Example 10-5 shows the output of the **show bgp peers** command for router R1 at this stage of the configuration.

Example 10-5 Verifying eBGP on R1: “show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.22        100    ESTAB     4    22         38       0         6
10.0.0.33        100    ESTAB     4    26         25       0         6
10.0.0.44        100    ESTAB     4    16         15       22        6
88.88.88.2       200    ESTAB     4    6          2        5         1
```

```
vyatta@R1>
```

After adding the eBGP peer 88.88.88.2 (the BGP ID configured for the router connected to AS 200) to R1 we can see that the new peer is in the ESTAB state. This indicates that the peer was properly preconfigured for this connection.

Additionally you may notice that the **Update Rx** field for peer 88.88.88.2 shows “5”. This shows that the peer has sent router R1 five BGP update messages, which are typically associated with a route announcement or withdrawal. Also, you can see that router R1 has transmitted one BGP update message to each of its iBGP peers.

R1: show bgp neighbor-routes

Example 10-6 shows the output of the **show bgp neighbor-routes** command for router R1 at this stage of the configuration.

Example 10-6 Verifying eBGP on R1: “show bgp neighbor-routes”

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                      Peer          AS Path
------                -------                      ----          -------
*> 2.0.0.0/24           88.88.88.2                   88.88.88.88   200 i
*> 2.1.0.0/24           88.88.88.2                   88.88.88.88   200 i
*> 2.2.0.0/24           88.88.88.2                   88.88.88.88   200 i
*> 2.0.0.0/8            88.88.88.2                88.88.88.88    200 i
*> 3.0.0.0/8            99.99.99.2                   10.0.0.44     300 i
*> 3.0.0.0/24           99.99.99.2                   10.0.0.44     300 i
*> 3.1.0.0/24           99.99.99.2                   10.0.0.44     300 i
*> 3.2.0.0/8            99.99.99.2                   10.0.0.44     300 i
*> 10.1.0.0/16          99.99.99.2                   10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2                   88.88.88.88   200 i
*> 172.16.128.0/18      99.99.99.2                   10.0.0.44     300 i
*> 192.168.2.0/24       99.99.99.2                   10.0.0.44     300 i
vyatta@R1>
```

From this output we can see that peer 88.88.88.88 has sent router R1 five prefix announcements, and that router R1 has learned seven prefixes from its iBGP peer R4. We can tell from the AS Path that some of these routes were originated by AS 200 and others were originated by AS 300.

There are two symbols that are displayed at the beginning of each of the prefixes shown in the output of **show bgp routes**. The first symbol is the status code for a valid route, which is an asterisk (“*”). Essentially all routes shown in the BGP table should be preceded by this symbol. The second symbol is the greater than character (“>”), which indicates which path is the best available path as determined by the BGP best path selection process. The **show bgp routes** command shows only the best path to each peer.

R1: show route protocol bgp

Example 10-7 shows the output of the **show route protocol bgp** command for router R1 at this stage of the configuration.

Example 10-7 Verifying eBGP on R1: “show route protocol bgp”

```
vyatta@R1> show route protocol bgp
Routes: 12/20, Paths: 12/20
2.0.0.0/8             [ebgp(0)]      > to 88.88.88.2          via eth0
2.0.0.0/24            [ebgp(0)]      > to 88.88.88.2          via eth0
2.1.0.0/24            [ebgp(0)]      > to 88.88.88.2          via eth0
2.2.0.0/24            [ebgp(0)]      > to 88.88.88.2          via eth0
3.0.0.0/8             [ibgp(0)]      > to 172.16.0.2          via eth1
3.0.0.0/24            [ibgp(0)]      > to 172.16.0.2          via eth1
3.1.0.0/24            [ibgp(0)]      > to 172.16.0.2          via eth1
3.2.0.0/24            [ibgp(0)]      > to 172.16.0.2          via eth1
10.1.0.0/16           [ibgp(0)]      > to 172.16.0.2          via eth1
12.0.0.0/8            [ebgp(0)]      > to 88.88.88.2          via eth0
172.16.128.0/18       [ibgp(0)]      > to 172.16.0.2          via eth1
192.168.2.0/24        [ibgp(0)]      > to 172.16.0.2          via eth1

vyatta@R1>
```

The **show route protocol bgp** command displays the routes in the RIB that were learned via BGP. This is different from the output of **show bgp routes**, which shows all paths learned via BGP regardless of whether it is the best BGP path and whether the BGP candidate to the RIB for the prefix is the best route (for example, it has the lowest admin cost).

The output for the same operational BGP commands run on router R4 yields similar results.

R4: show bgp peers

Example 10-8 shows the output of the **show bgp peers** command for router R4 at this stage of the configuration.

Example 10-8 Verifying eBGP on R4: “show bgp peers”

```
vyatta@R4> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.11        100    ESTAB     4    66         62       12        5
10.0.0.22        100    ESTAB     4    54         62       0         5
10.0.0.33        100    ESTAB     4    57         61       0         5
99.99.99.2       300    ESTAB     4    10         5        7         1

vyatta@R4>
```

R4: show bgp neighbor-routes

Example 10-9 shows the output of the **show bgp neighbor-routes** command for router R4 at this stage of the configuration.

Example 10-9 Verifying eBGP on R4: “show bgp neighbor-routes”

```
vyatta@R4> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                    Peer          AS Path
------                -------                    ----          -------
*> 2.0.0.0/24           88.88.88.2                 10.0.0.11     200 i
*> 2.1.0.0/24           88.88.88.2                 10.0.0.11     200 i
*> 2.2.0.0/24           88.88.88.2                 10.0.0.11     200 i
*> 2.0.0.0/8            88.88.88.2                 10.0.0.11     200 i
*> 3.0.0.0/8            99.99.99.2                 99.99.99.99   300 i
*> 3.0.0.0/24           99.99.99.2                 99.99.99.99   300 i
*> 3.1.0.0/24           99.99.99.2                 99.99.99.99   300 i
*> 3.2.0.0/8            99.99.99.2                 99.99.99.99   300 i
*> 10.1.0.0/16          99.99.99.2                 99.99.99.99   300 i
*> 12.0.0.0/8           88.88.88.2                 10.0.0.11     200 i
*> 172.16.128.0/18      99.99.99.2                 99.99.99.99   300 i
*> 192.168.2.0/24       99.99.99.2                 99.99.99.99   300 i
vyatta@R4>
```

Router R4’s BGP table contains the paths it learned from its eBGP peer, as well as the paths it learned from its iBGP neighbor R1.

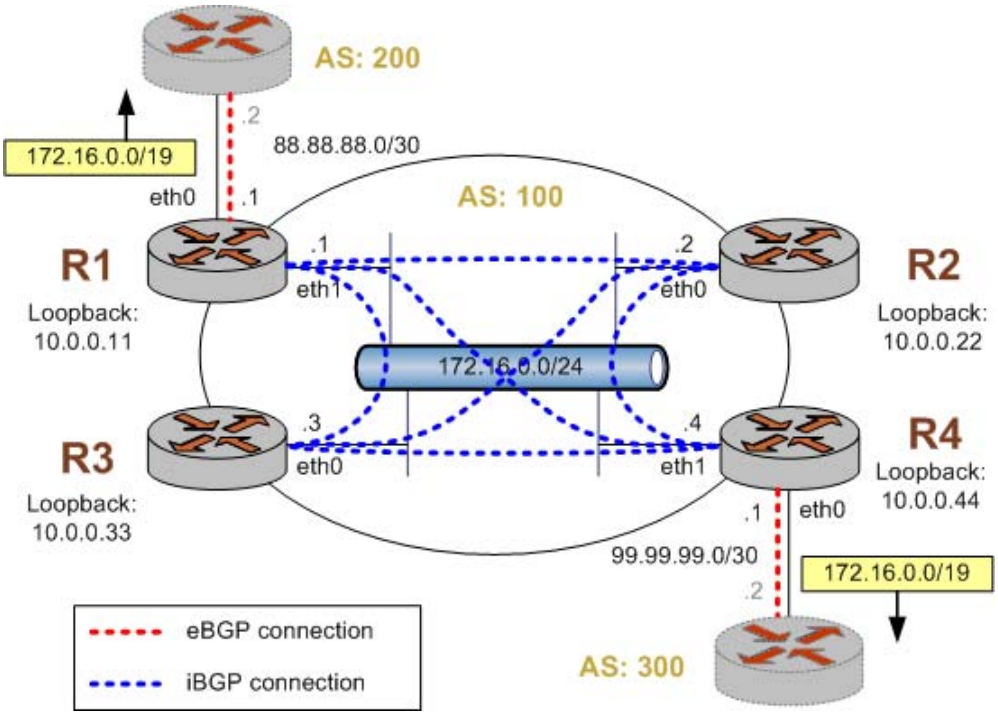
Originating a Route to eBGP Neighbors

One of the common requirements for BGP configurations is to originate a network prefix to BGP peers. On the Vyatta router this is accomplished using a combination of static routes and routing policies.

In this section, you originate the network prefix from both the R1 and R4 routers. This is shown in Figure 10-8.

NOTE *The example in this section assumes that the desired network to originate to our BGP peers is 172.16.0.0/19, which is a private RFC 1918 network address. Typically, the router would be originating a public IP network assigned by an Internet routing registry such as the American Registry for Internet Numbers (ARIN).*

Figure 10-8 Originating a route to eBGP neighbors



This example assumes that the configurations in previous sections have been performed.

To originate a route to eBGP neighbors, perform the following steps in configuration mode:

Example 10-10 Originating routes to eBGP neighbors

Router	Step	Command(s)
R1	Create a static route entry for 172.16.0.0/19 with a next hop of the loopback IP address.	vyatta@R1# **set protocols static route 172.16.0.0/19 next-hop 10.0.0.11** [edit]
R1	Create a routing policy to redistribute this specific static route.	vyatta@R1# **set policy policy-statement BGP_EXPORT term 1 from network4 172.16.0.0/19** [edit] vyatta@R1# **set policy policy-statement BGP_EXPORT term 1 from protocol static** [edit] vyatta@R1# **set policy policy-statement BGP_EXPORT term 1 then action accept** [edit]
R1	Apply the routing policy to the BGP protocol.	vyatta@R1# **set protocols bgp export BGP_EXPORT** [edit]
R1	Commit the configuration.	vyatta@R1# **commit** OK [edit]
R4	Create a static route entry for 172.16.0.0/19 with a next hop of loopback IP address.	vyatta@R4# **set protocols static route 172.16.0.0/19 next-hop 10.0.0.44** [edit]
R4	Create a routing policy to redistribute this specific static route.	vyatta@R4# **set policy policy-statement BGP_EXPORT term 1 from network4 172.16.0.0/19** [edit] vyatta@R4# **set policy policy-statement BGP_EXPORT term 1 from protocol static** [edit] vyatta@R4# **set policy policy-statement BGP_EXPORT term 1 then action accept** [edit]
R4	Apply the routing policy to the BGP protocol	vyatta@R4# **set protocols bgp export BGP_EXPORT** [edit]

Example 10-10 Originating routes to eBGP neighbors

R4	Commit the configuration.	`vyatta@R4# commit`
		`OK`
		`[edit]`

Verifying the Route Origination

The following commands can be used to verify the route origination configuration.

R1: show bgp peers

Example 10-11 shows the output of the **show bgp peers** command for router R1 at this stage of the configuration. The Update Tx column indicates that the router has been sending BGP updates, showing the number of BGP update messages that have been sent to each neighbor.

Example 10-11 Verifying route origination on R1: “show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.22        100    ESTAB     4    22         38       0         10
10.0.0.33        100    ESTAB     4    26         25       0         10
10.0.0.44        100    ESTAB     4    16         15       0         10
88.88.88.2       100    ESTAB     4    6          2        1         1

vyatta@R1>
```

R1: show bgp neighbor-routes

Example 10-12 shows the output of the **show bgp neighbor-routes** command for router R1 at this stage of the configuration.The AS column shows the AS path. You can check this column to make sure you received the route that you expected from each BGP neighbor.

Example 10-12 Verifying route origination on R1: “show bgp neighbor-routes”

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                 Peer          AS Path
------                -------                 ----          -------
*> 2.0.0.0/24           88.88.88.2            88.88.88.88   200 i
*> 2.1.0.0/24           88.88.88.2            88.88.88.88   200 i
*> 2.2.0.0/24           88.88.88.2            88.88.88.88   200 i
*> 2.0.0.0/8            88.88.88.2            88.88.88.88   200 i
*> 3.0.0.0/8            99.99.99.2            10.0.0.44     300 i
*> 3.0.0.0/24           99.99.99.2            10.0.0.44     300 i
*> 3.1.0.0/24           99.99.99.2            10.0.0.44     300 i
*> 3.2.0.0/8            99.99.99.2            10.0.0.44     300 i
*> 10.1.0.0/16          99.99.99.2            10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2            88.88.88.88   200 i
*> 172.16.0.0/19        10.0.0.11             0.0.0.0       i
*  172.16.0.0/19        10.0.0.44             10.0.0.44     i
*> 172.16.128.0/18      99.99.99.2            10.0.0.44     300 i
*> 192.168.2.0/24       99.99.99.2            10.0.0.44     300 i
vyatta@R1>
```

Router R1 shows the prefix 172.16.0.0/19 from peer 0.0.0.0 which is used to represent itself and it also shows the 172.16.0.0/19 prefix being learned from iBGP neighbor R4. Note that the AS Path for locally originated routes does not contain any AS numbers in the AS Path, this is because the originating AS (AS 100) will be added to BGP updates sent to eBGP peers.

This is also the first case where we have more than one path available for a prefix (172.16.0.0/19) since it is being originated on both R1 and R4. In this case, on R1, the route originated on R1 is the best path and has been selected as such by the BGP best path selection process. We can see that there is a second path available via R4. If for some reason the locally originated route is withdrawn from the BGP table, then the path via R4 would become the best path available.

AS 200: show bgp neighbor-routes

Example 10-13 shows the output of the **show bgp neighbor-routes** command for the router in AS 200 at this stage of the configuration.

Example 10-13 Verifying route origination in AS 200: “show bgp neighbor-routes”

```
vyatta@AS200> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                 Peer          AS Path
------                -------                 ----          -------
*> 2.0.0.0/8           88.88.88.88              0.0.0.0       i
*> 2.0.0.0/24          88.88.88.88              0.0.0.0       i
*> 2.1.0.0/24          88.88.88.88              0.0.0.0       i
*> 2.2.0.0/24          88.88.88.88              0.0.0.0       i
*> 3.0.0.0/8           88.88.88.1               10.0.0.11     100 300 i
*> 3.0.0.0/24          88.88.88.1               10.0.0.11     100 300 i
*> 3.1.0.0/24          88.88.88.1               10.0.0.11     100 300 i
*> 3.2.0.0/24          88.88.88.1               10.0.0.11     100 300 i
*> 10.1.0.0/16         88.88.88.1               10.0.0.11     100 300 i
*> 12.0.0.0/8          88.88.88.88              0.0.0.0       i
*> 172.16.0.0/19       88.88.88.1               10.0.0.11     100 i
*> 172.16.128.0/18     88.88.88.1               10.0.0.11     100 300 i
*> 192.168.2.0/24      88.88.88.1               10.0.0.11     100 300 i
vyatta@AS200>
```

We can confirm that router R1’s peer is learning the route via BGP by checking the BGP table on that router. Note that normally you would not be able to do this, since the router is in a different AS and therefore is probably controlled by a different organization. The 172.16.0.0/19 prefix is being learned from peer 10.0.0.11 (R1), is reachable via the next-hop 88.88.88.1, and has an origin AS of 100.

Inbound Route Filtering

Another common requirement for BGP configurations is to filter inbound routing announcements from a BGP peer. On the Vyatta router this is accomplished using routing policies that are then applied to the BGP process as “import” policies.

Example 10-14 creates the following inbound filtering policies:

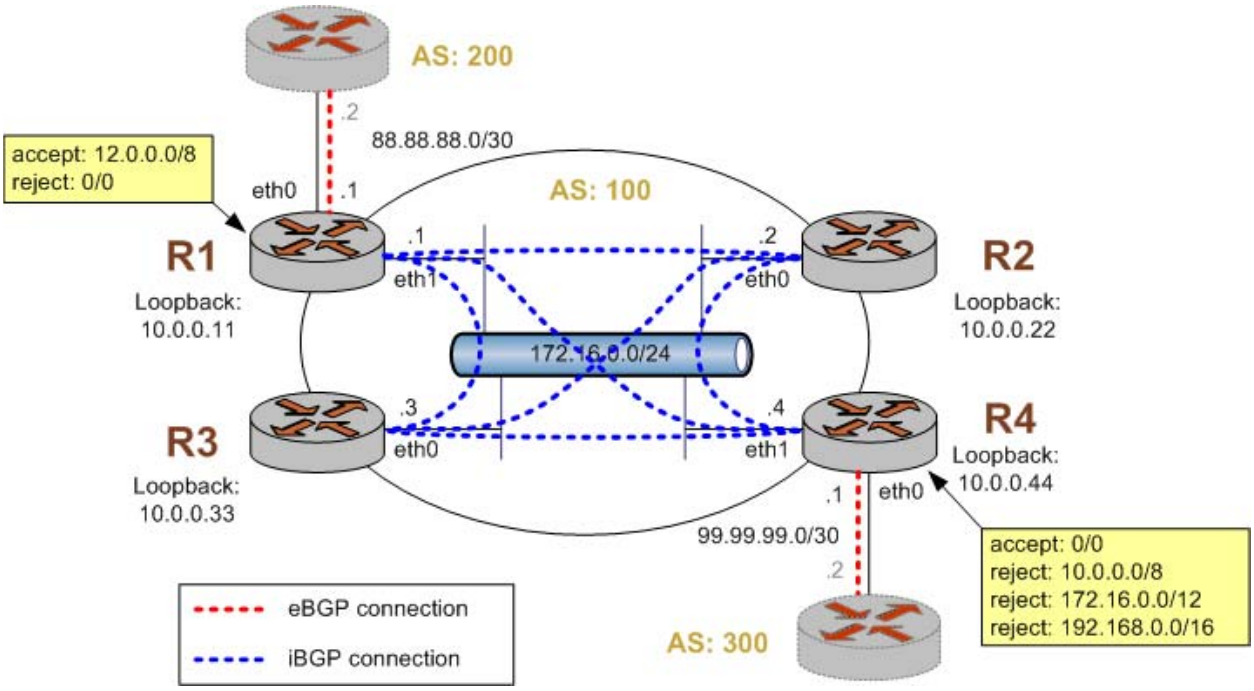
- R1 should only accept network 12.0.0.0/8 from its eBGP peer, and reject everything else.

- R4 should allow all Internet routes, but reject all RFC 1918 networks from its eBGP peer.

This import policy is shown in Figure 10-9.

NOTE *Because the routing policy is applied to the BGP process, by default it applies to all BGP peers. By defining the peer that the route is being announced by, we can restrict the policy to only affect the desired peer.*

Figure 10-9 Filtering inbound routes



Note that the Vyatta device does not currently allow a routing policy to be applied to a specific BGP peer; the policy is applied globally to the BGP process. This means that to avoid having a policy action affect all peers, if the action should be restricted to a single peer the peer should be identified in the policy term using the **neighbor** keyword.

To create this inbound route filter, perform the following steps in configuration mode:

Example 10-14 Creating an import policy

Router	Step	Command(s)
R1	Create a routing policy to filter out inbound route announcements from R1's eBGP peer.	vyatta@R1# **set policy policy-statement R1_IMPORT_FILTER term 1 from network4 12.0.0.0/8** [edit] vyatta@R1# **set policy policy-statement R1_IMPORT_FILTER term 1 from neighbor 88.88.88.88** [edit]
R1		vyatta@R1# **set policy policy-statement R1_IMPORT_FILTER term 1 then action accept** [edit] vyatta@R1# **set policy policy-statement R1_IMPORT_FILTER term 2 from neighbor 88.88.88.88** [edit] vyatta@R1# **set policy policy-statement R1_IMPORT_FILTER term 2 then action reject** [edit]
R1	Apply the routing policy to the BGP protocol.	vyatta@R1# **set protocols bgp import R1_IMPORT_FILTER** [edit]
R1	Commit the configuration.	vyatta@R1# **commit** OK [edit]
R4	Create a routing policy to filter out inbound route announcements from R4's eBGP peer.	vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 1 from network4 orlonger 10.0.0.0/8** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 1 from neighbor 99.99.99.99** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 1 then action reject** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 2 from network4 orlonger 172.16.0.0/12**

Example 10-14 Creating an import policy

R4		vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 2 from neighbor 99.99.99.99** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 2 then action reject** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 3 from network4 orlonger 192.168.0.0/16** [edit]
R4		vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 3 from neighbor 99.99.99.99** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 3 then action reject** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 3 from neighbor 99.99.99.2** [edit] vyatta@R4# **set policy policy-statement R4_IMPORT_FILTER term 3 then action accept** [edit]
R4	Apply the routing policy to the BGP protocol	vyatta@R4# **set protocols bgp import R4_IMPORT_FILTER** [edit]
R4	Commit the configuration.	vyatta@R4# **commit** OK [edit]

Verifying the Inbound Filter

The following commands can be used to verify the inbound filter configuration.

R1: show bgp neighbor-routes

Example 10-15 shows R1's BGP table *before* the import filter is applied.

Example 10-15 R1 inbound BGP routes before import filtering

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                 Peer          AS Path
------                -------                 ----          -------
*> 2.0.0.0/24           88.88.88.2              88.88.88.88   200 i
*> 2.1.0.0/24           88.88.88.2              88.88.88.88   200 i
*> 2.2.0.0/24           88.88.88.2              88.88.88.88   200 i
*> 2.0.0.0/8            88.88.88.2              88.88.88.88   200 i
*> 3.0.0.0/8            99.99.99.2              10.0.0.44     300 i
*> 3.0.0.0/24           99.99.99.2              10.0.0.44     300 i
*> 3.1.0.0/24           99.99.99.2              10.0.0.44     300 i
*> 3.2.0.0/8            99.99.99.2              10.0.0.44     300 i
*> 10.1.0.0/16          99.99.99.2              10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2              88.88.88.88   200 i
*> 172.16.0.0/19        10.0.0.11               0.0.0.0       i
*  172.16.0.0/19        10.0.0.44               10.0.0.44     i
*> 172.16.128.0/18      99.99.99.2              10.0.0.44     300 i
*> 192.168.2.0/24       99.99.99.2              10.0.0.44     300 i
vyatta@R1>
```

R1: show bgp neighbor-routes

Example 10-16 shows R1's BGP table *after* the import filter is applied.

Example 10-16 R1 inbound BGP routes after import filtering

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                 Peer          AS Path
------                -------                 ----          -------
```

```
*  2.0.0.0/24           88.88.88.2                88.88.88.88   200 i
*  2.1.0.0/24           88.88.88.2                88.88.88.88   200 i
*  2.2.0.0/24           88.88.88.2                88.88.88.88   200 i
*  2.0.0.0/8            88.88.88.2                88.88.88.88   200 i
*> 3.0.0.0/8            99.99.99.2                10.0.0.44     300 i
*> 3.0.0.0/24           99.99.99.2                10.0.0.44     300 i
*> 3.1.0.0/24           99.99.99.2                10.0.0.44     300 i
*> 3.2.0.0/8            99.99.99.2                10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2                88.88.88.88   200 i
*> 172.16.0.0/19        10.0.0.11                 0.0.0.0       i
*  172.16.0.0/19        10.0.0.44                 10.0.0.44     i
vyatta@R1>
```

Note that while the filtered prefixes from peers 88.88.88.88 are still in the BGP table, the ">" marker indicating that the path is the best path has been removed from the filtered prefixes. This also means that these filtered paths will not be sent to any of R1's eBGP or iBGP peers.

R4: show bgp neighbor-routes

Example 10-17 shows R4's BGP table *before* the import filter is applied.

Example 10-17 R4 inbound BGP routes before import filtering

```
vyatta@R4> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix               Nexthop                   Peer          AS Path
------               -------                   ----          -------
*> 2.0.0.0/24          88.88.88.2                10.0.0.11     200 i
*> 2.1.0.0/24          88.88.88.2                10.0.0.11     200 i
*> 2.2.0.0/24          88.88.88.2                10.0.0.11     200 i
*> 2.0.0.0/8           88.88.88.2                10.0.0.11     200 i
*> 3.0.0.0/8           99.99.99.2                99.99.99.99   300 i
*> 3.0.0.0/24          99.99.99.2                99.99.99.99   300 i
*> 3.1.0.0/24          99.99.99.2                99.99.99.99   300 i
*> 3.2.0.0/8           99.99.99.2                99.99.99.99   300 i
*> 10.1.0.0/16         99.99.99.2                99.99.99.99   300 i
*> 12.0.0.0/8          88.88.88.2                10.0.0.11     200 i
*> 172.16.0.0/19       10.0.0.44                 0.0.0.0       i
*  172.16.0.0/19       10.0.0.11                 10.0.0.11     i
*> 172.16.128.0/18     99.99.99.2                99.99.99.99   300 i
*> 192.168.2.0/24      99.99.99.2                99.99.99.99   300 i
vyatta@R4>
```

R4: show bgp neighbor-routes

The output below shows R4's BGP table *after* the import filter is applied.

Example 10-18 R4 inbound BGP routes after import filtering

```
vyatta@R4> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                    Peer          AS Path
------                -------                    ----          -------
*> 3.0.0.0/8           99.99.99.2                  99.99.99.99   300 i
*> 3.0.0.0/24          99.99.99.2                  99.99.99.99   300 i
*> 3.1.0.0/24          99.99.99.2                  99.99.99.99   300 i
*> 3.2.0.0/8           99.99.99.2                  99.99.99.99   300 i
*  10.1.0.0/16         99.99.99.2                  99.99.99.99   300 i
*> 12.0.0.0/8          88.88.88.2                  10.0.0.11     200 i
*> 172.16.0.0/19       10.0.0.44                   0.0.0.0       i
*  172.16.0.0/19       10.0.0.11                   10.0.0.11     i
*  172.16.128.0/18     99.99.99.2                  99.99.99.99   300 i
*  192.168.2.0/24      99.99.99.2                  99.99.99.99   300 i
vyatta@R4>
```

Outbound Route Filtering

Filtering outbound prefixes is another common BGP configuration requirement. On the Vyatta router this is accomplished using routing policies that are then applied to the BGP process as "export" policies.

The example in this section assumes that AS100 does not want to be a transit AS for AS 200 or AS 300. This means that:

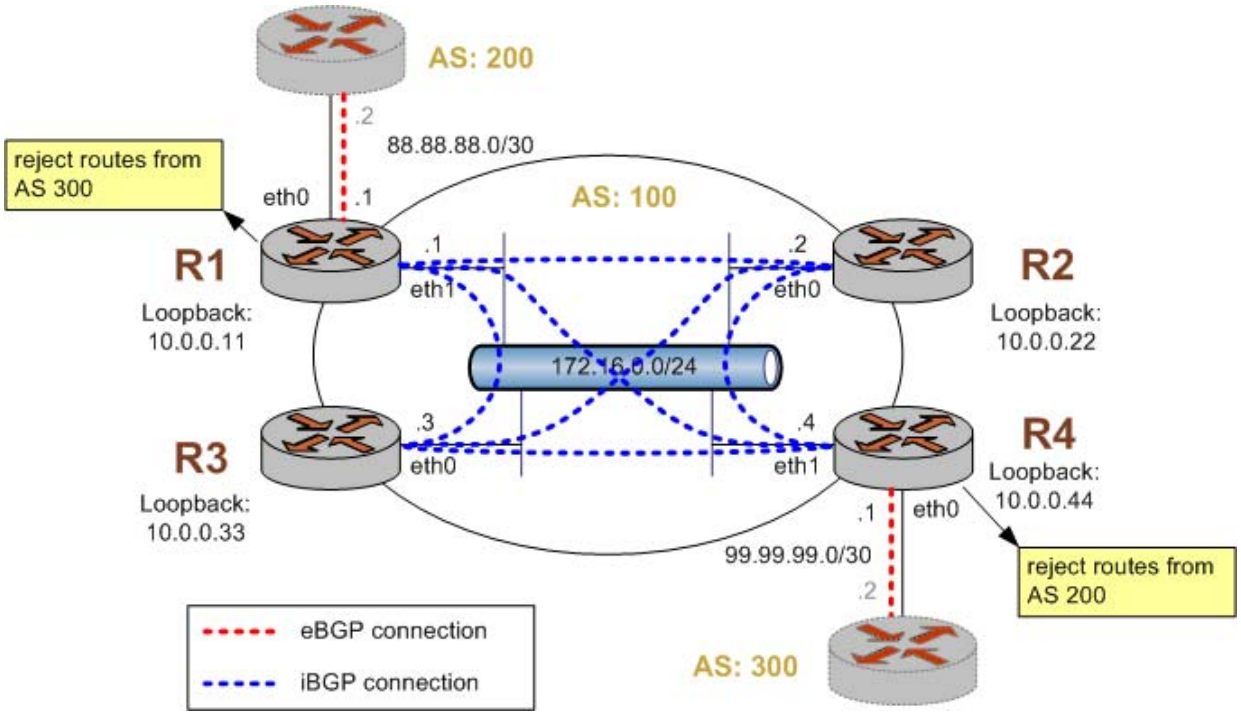
- eBGP routes from R1's eBGP peer (AS 200) should not be sent to R4's eBGP peer.
- Routes from R4's eBGP peer (AS 300) should not be sent to R1's eBGP peer.

If we *did not* implement this filtering, AS 300 might send traffic destined for AS 200 to router R4, and this traffic would then be carried across the AS 100 network.

There are several ways that this routing policy could be implemented: two most common are basing the filter on the network prefix or basing it on the AS Path. In this example, we update the existing BGP export policy to add some additional restrictions that will prevent AS 100 from acting as a transit network for AS 200 and AS 300.

This export policy is shown in Figure 10-10.

Figure 10-10 Filtering outbound routes



Note that the Vyatta device does not currently allow a routing policy to be applied to a specific BGP peer; instead, the policy is applied globally to the BGP process. This is required to avoid having a policy action affect all peers. If the action should be restricted to a single peer, the peer should be identified in the policy term using the **neighbor** keyword.

To create this export policy, perform the following steps in configuration mode:

Example 10-19 Creating an export policy

Router	Step	Command(s)

R1	Add a second term to the policy to filter out all networks from being sent to the eBGP peer.	`vyatta@R1# set policy policy-statement BGP_EXPORT term 2 from network4 orlonger 0.0.0.0/0`
		`[edit]`
		`vyatta@R1# set policy policy-statement BGP_EXPORT term 2 from protocol bgp`
		`[edit]`

Example 10-19 Creating an export policy

R1		vyatta@R1# **set policy policy-statement BGP_EXPORT term 2 to neighbor 88.88.88.88** [edit] vyatta@R1# **set policy policy-statement BGP_EXPORT term 2 then action reject** [edit]
R1	Commit the configuration.	vyatta@R1# **commit** OK [edit]
R4	Add a second term to the policy to filter out all networks from being sent to the eBGP peer.	vyatta@R4# **set policy policy-statement BGP_EXPORT term 2 from network4 orlonger 0.0.0.0/0** [edit] vyatta@R4# **set policy policy-statement BGP_EXPORT term 2 from protocol bgp** [edit] vyatta@R4# **set policy policy-statement BGP_EXPORT term 2 to neighbor 99.99.99.99** [edit] vyatta@R4# **set policy policy-statement BGP_EXPORT term 2 then action reject** [edit]
R4	Commit the configuration.	vyatta@R4# **commit** OK [edit]

Verifying the Outbound Filter

The following commands can be used to verify the outbound filter configuration.

AS 200: show bgp neighbor-routes

Example 10-20 shows AS 200's BGP table *before* the import filter is applied.

Example 10-20 AS 200 outbound BGP routes before export filtering

```
vyatta@AS200> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                 Peer          AS Path
```

```
------                  -------                       ----          -------
*> 2.0.0.0/8            88.88.88.88                     0.0.0.0       i
*> 2.0.0.0/24           88.88.88.88                     0.0.0.0       i
*> 2.1.0.0/24           88.88.88.88                     0.0.0.0       i
*> 2.2.0.0/24           88.88.88.88                     0.0.0.0       i
*> 3.0.0.0/8            88.88.88.1                      10.0.0.11     100 300 i
*> 3.0.0.0/24           88.88.88.1                      10.0.0.11     100 300 i
*> 3.1.0.0/24           88.88.88.1                      10.0.0.11     100 300 i
*> 3.2.0.0/24           88.88.88.1                      10.0.0.11     100 300 i
*> 10.1.0.0/16          88.88.88.1                      10.0.0.11     100 300 i
*> 12.0.0.0/8           88.88.88.88                     0.0.0.0       i
*> 172.16.0.0/19        88.88.88.1                      10.0.0.11     100 i
*> 172.16.128.0/18      88.88.88.1                      10.0.0.11     100 300 i
*> 192.168.2.0/24       88.88.88.1                      10.0.0.11     100 300 i
vyatta@AS200>
```

AS 200: show bgp neighbor-routes

Example 10-21 shows AS 200's BGP table *after* the import filter is applied.

Example 10-21 AS 200 outbound BGP routes after export filtering

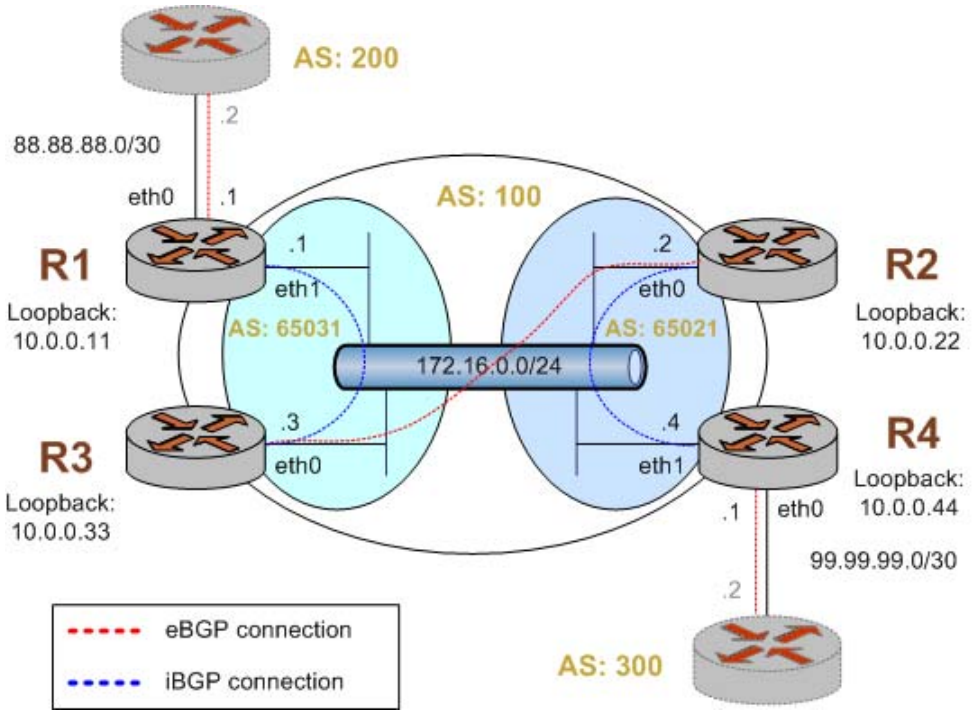
```
vyatta@AS200> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                  Nexthop                       Peer          AS Path
------                  -------                       ----          -------
*> 2.0.0.0/8            88.88.88.88                     0.0.0.0       i
*> 2.0.0.0/24           88.88.88.88                     0.0.0.0       i
*> 2.1.0.0/24           88.88.88.88                     0.0.0.0       i
*> 2.2.0.0/24           88.88.88.88                     0.0.0.0       i
*> 12.0.0.0/8           88.88.88.88                     0.0.0.0       i
*> 172.16.0.0/19        88.88.88.1                      10.0.0.11     100 i
vyatta@AS200>
```

Confederations

Confederations allow large Autonomous Systems to sub-divide the AS into sub-ASs. This helps solve the scalability issues associated with having to maintain a full mesh of iBGP connections between all iBGP routers in the AS. In the confederation example shown in Figure 10-11, routers R1 and R3 are configured in one sub-AS (AS number 65031) and routers R2 and R4 are configured in a different sub-AS (AS number 65021).

Figure 10-11 BGP confederation



This example assumes that the configurations in previous sections have been performed.

To create the confederation shown in Figure 10-11, perform the following steps in configuration mode

Example 10-22 Creating a BGP confederation

Router	Step	Command(s)
R1	Delete current iBGP peer configurations.	vyatta@R1# **delete protocols bgp peer 10.0.0.22** [edit] vyatta@R1# **delete protocols bgp peer 10.0.0.33** [edit] vyatta@R1# **delete protocols bgp peer 10.0.0.44** [edit]
R1	Change local AS to match the confederation sub-AS.	vyatta@R1# **set protocols bgp local-as 65031** [edit]
R1	Add the true AS that the router belongs to as the confederation identifier.	vyatta@R1# **set protocols bgp confederation identifier 100** [edit]
R1	Add an iBGP peer connection to R3.	vyatta@R1# **set protocols bgp peer 10.0.0.33 as 65031** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 confederation-member true** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 local-ip 10.0.0.11** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 next-hop 10.0.0.11** [edit]
R1	Commit the configuration.	vyatta@R1# **commit** OK [edit]
R2	Delete current iBGP peer configurations.	vyatta@R2# **delete protocols bgp peer 10.0.0.11** [edit] vyatta@R2# **delete protocols bgp peer 10.0.0.33** [edit] vyatta@R2# **delete protocols bgp peer 10.0.0.44** [edit]

Example 10-22 Creating a BGP confederation

R2	Change the local AS to match the confederation sub-AS.	vyatta@R2# **set protocols bgp local-as 65021** [edit]
R2	Add the true AS that the router belongs to as the confederation identifier.	vyatta@R2# **set protocols bgp confederation identifier 100** [edit]
R2	Add the iBGP peer connection to R4.	vyatta@R2# **set protocols bgp peer 10.0.0.44 as 65021** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.44 confederation-member true** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.44 local-ip 10.0.0.22** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.44 next-hop 10.0.0.22** [edit]
R2	Add the eBGP peer connection to R3.	vyatta@R2# **set protocols bgp peer 10.0.0.33 as 65031** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.33 confederation-member true** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.33 local-ip 10.0.0.22** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.33 next-hop 10.0.0.22** [edit]
R2	Commit the configuration.	vyatta@R2# **commit** OK [edit]
R3	Delete current iBGP peer configurations.	vyatta@R3# **delete protocols bgp peer 10.0.0.11** [edit] vyatta@R3# **delete protocols bgp peer 10.0.0.22** [edit] vyatta@R3# **delete protocols bgp peer 10.0.0.44** [edit]

Example 10-22 Creating a BGP confederation

R3	Change the local AS to match the confederation sub-AS.	vyatta@R3# **set protocols bgp local-as 65031** [edit]
R3	Add the true AS that the router belongs to as the confederation identifier	vyatta@R3# **set protocols bgp confederation identifier 100** [edit]
R3	Add the iBGP peer connection to R1.	vyatta@R3# **set protocols bgp peer 10.0.0.11 as 65031** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.11 confederation-member true** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.11 local-ip 10.0.0.33** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.11 next-hop 10.0.0.33** [edit]
R3	Add the eBGP peer connection to R2.	vyatta@R3# **set protocols bgp peer 10.0.0.22 as 65021** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.22 confederation-member true** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.22 local-ip 10.0.0.22** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.22 next-hop 10.0.0.22** [edit]
R3	Commit the configuration.	vyatta@R3# **commit** OK [edit]
R4	Delete current iBGP peer configurations.	vyatta@R4# **delete protocols bgp peer 10.0.0.11** [edit] vyatta@R4# **delete protocols bgp peer 10.0.0.22** [edit] vyatta@R4# **delete protocols bgp peer 10.0.0.33** [edit]

Example 10-22 Creating a BGP confederation

R4	Change the local AS to match the confederation sub-AS.	vyatta@R4# **set protocols bgp local-as 65021** [edit]
R4	Add the true AS that the router belongs to as the confederation identifier.	vyatta@R4# **set protocols bgp confederation identifier 100** [edit]
R4	Add the iBGP peer connection to R2.	vyatta@R4# **set protocols bgp peer 10.0.0.22 as 65021** [edit] vyatta@R4# **set protocols bgp peer 10.0.0.22 confederation-member true** [edit] vyatta@R4# **set protocols bgp peer 10.0.0.22 local-ip 10.0.0.44** [edit] vyatta@R4# **set protocols bgp peer 10.0.0.22 next-hop 10.0.0.44** [edit]
R4	Commit the configuration.	vyatta@R4# **commit** OK [edit]

Verifying the Confederation

The following commands can be used to verify the confederation configuration.

R1: show bgp peers

Example 10-23 shows the output of the **show bgp peers** command for router R1 at this stage of the configuration.

Example 10-23 Verifying confederations on R1: “show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------      -------  --------  ---------
10.0.0.33        65031  ESTAB     4    1879        1903     9         6
88.88.88.2       200    ESTAB     4    2569        2594     5         10

vyatta@R1>
```

R1: show bgp neighbor-routes

Example 10-24 shows the output of the **show bgp neighbor-routes** command for router R1 at this stage of the configuration.

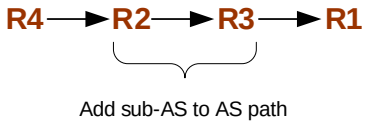
Example 10-24 Verifying confederations on R1: “show bgp neighbor-routes”

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix               Nexthop                    Peer          AS Path
------               -------                    ----          -------
*> 172.16.0.0/19       10.0.0.11                  0.0.0.0       i
*> 2.0.0.0/24          99.99.99.2                 10.0.0.33     (65021) 300 i
*> 2.1.0.0/24          99.99.99.2                 10.0.0.33     (65021) 300 i
*> 2.2.0.0/24          99.99.99.2                 10.0.0.33     (65021) 300 i
*> 2.0.0.0/8           99.99.99.2                 10.0.0.33     (65021) 300 i
*> 12.0.0.0/8          88.88.88.2                 88.88.88.2    200 i
vyatta@R1>
```

Note that the routes learned from router R3 (peer 10.0.0.33) include the confederation sub-AS in the AS Path. All confederation sub-ASs will be shown inside brackets (). This information is not transmitted outside of the true AS (AS 100).

Example 10-25 shows R2's sample output for the AS Path of prefix 3.0.0.0/8 as it is announced between the iBGP peers with AS 100, as follows:



R2: show bgp peers

Example 10-25 shows the output of the **show bgp peers** command for router R2 at this stage of the configuration.

Example 10-25 Verifying confederations on R2: "show bgp peers"

```
vyatta@R2> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------      -------  --------  ---------
10.0.0.33        65031  ESTAB     4    1867        2821     6         18
10.0.0.44        65021  ESTAB     4    588         931      8         11

vyatta@R1>
```

R2: show bgp neighbor-routes

Example 10-26 shows the output of the **show bgp neighbor-routes** command for router R2 at this stage of the configuration.

Example 10-26 Verifying confederations on R2: "show bgp neighbor-routes"

```
vyatta@R2> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix               Nexthop                  Peer          AS Path
------               -------                  ----          -------
*> 2.0.0.0/24          99.99.99.2               10.0.0.33     300 i
*> 2.1.0.0/24          99.99.99.2               10.0.0.33     300 i
```

```
*> 2.2.0.0/24           99.99.99.2                 10.0.0.33    300 i
*> 2.0.0.0/8            99.99.99.2                 10.0.0.33    300 i
*> 12.0.0.0/8           99.99.99.2                  10.0.0.33  (65031) 200 i
*  172.16.0.0/19        10.0.0.11                  10.0.0.33   (65031) i
*> 172.16.0.0/19        10.0.0.44                  10.0.0.44    i
vyatta@R2>
```

R3: show bgp peers

Example 10-27 shows the output of the **show bgp peers** command for router R3 at this stage of the configuration.

Example 10-27 Verifying confederations on R3: “show bgp peers”

```
vyatta@R3> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.11        65031  ESTAB     4    1956       1948     6         28
10.0.0.22        65021  ESTAB     4    72         38       17        6
vyatta@R3>
```

R3: show bgp neighbor-routes

Example 10-28 shows the output of the **show bgp neighbor-routes** command for router R3 at this stage of the configuration.

Example 10-28 Verifying confederations on R3: “show bgp neighbor-routes”

```
vyatta@R3> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix               Nexthop                    Peer          AS Path
------               -------                    ----          -------
*> 2.0.0.0/24          99.99.99.2                 10.0.0.22    (65021) 300 i
*> 2.1.0.0/24          99.99.99.2                 10.0.0.22    (65021) 300 i
*> 2.2.0.0/24          99.99.99.2                 10.0.0.22    (65021) 300 i
*> 2.0.0.0/8           99.99.99.2                 10.0.0.22    (65021) 300 i
*> 12.0.0.0/8          88.88.88.2                 10.0.0.33    200 i
```

```
*> 172.16.0.0/19        10.0.0.11                    10.0.0.11    i
*  172.16.0.0/19        10.0.0.44                    10.0.0.22    (65021) i
vyatta@R3>
```

R4: show bgp peers

Example 10-29 shows the output of the **show bgp peers** command for router R4 at this stage of the configuration.

Example 10-29 Verifying confederations on R4: “show bgp peers”

```
vyatta@R4> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.22        65021  ESTAB     4    76         48       7         8
99.99.99.2       300    ESTAB     4    1922       1933     7         39

vyatta@R4>
```

R4: show bgp neighbor-routes

Example 10-30 shows the output of the **show bgp neighbor-routes** command for router R4 at this stage of the configuration.

Example 10-30 Verifying confederations on R4: “show bgp neighbor-routes”

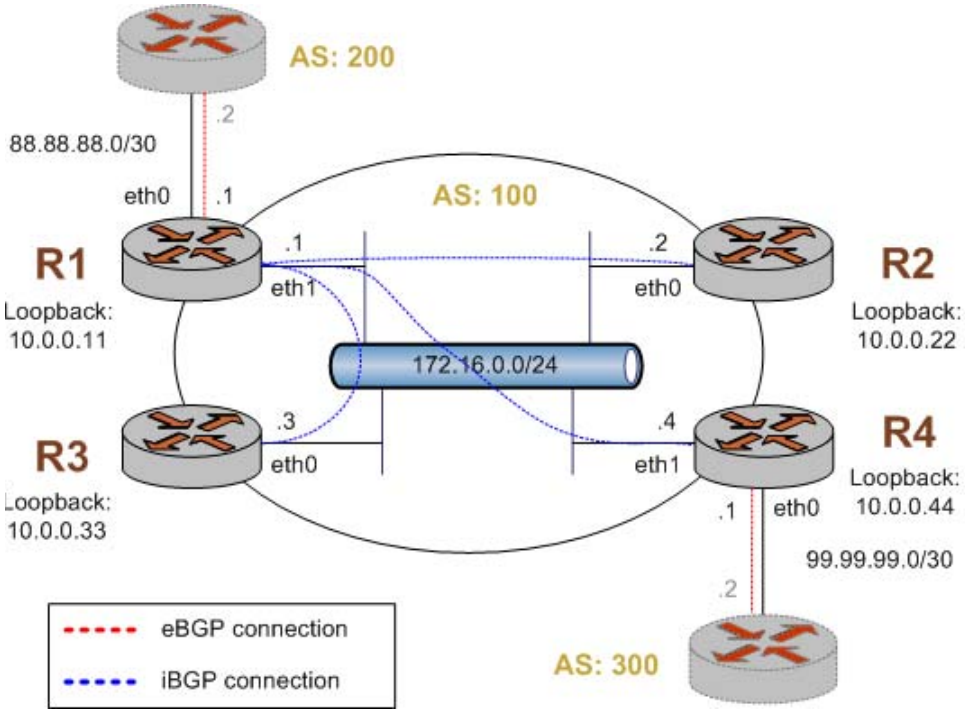
```
vyatta@R4> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix               Nexthop                    Peer          AS Path
------               -------                    ----          -------
*> 2.0.0.0/24          99.99.99.2                 99.99.99.99  300 i
*> 2.1.0.0/24          99.99.99.2                 99.99.99.99  300 i
*> 2.2.0.0/24          99.99.99.2                 99.99.99.99  300 i
*> 2.0.0.0/8           99.99.99.2                 99.99.99.99  300 i
*> 12.0.0.0/8          88.88.88.2                 10.0.0.22    (65031) 200 i
*> 172.16.0.0/19       10.0.0.44                  0.0.0.0      i
vyatta@R4>
```

Route Reflectors

Router reflectors are another technology designed to help BGP scale to large Autonomous Systems. In a route reflector configuration there is at least one route reflector server and one or more route reflector clients. In the example shown in Figure 10-12, router R1 is the route reflector server and router R2, R3, and R4 are the route reflector clients.

Figure 10-12 BGP route reflector



This example assumes that the configurations in previous sections have been performed, and that interfaces and OSPF have been configured. If you are starting from a clean base system you need not delete previous configuration.

Example 10-31 Creating route reflectors

Router	Step	Command(s)

R1	Delete confederation configurations and change the local-as back to 100.	`vyatta@R1# delete protocols bgp peer 10.0.0.33`
		`[edit]`
		`vyatta@R1# delete protocols bgp confederation`
		`[edit]`
		`vyatta@R1# set protocols bgp local-as 100`
		`[edit]`

Example 10-31 Creating route reflectors

R1	Set the route reflector cluster ID to 100.100.100.100	vyatta@R1# **set protocols bgp route-reflector cluster-id 100.100.100.100** [edit]
R1	Configure iBGP peering with R2.	vyatta@R1# **set protocols bgp peer 10.0.0.22 as 100** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.22 client true** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.22 local-ip 10.0.0.11** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.22 next-hop 10.0.0.11** [edit]
R1	Configure iBGP peering with R3.	vyatta@R1# **set protocols bgp peer 10.0.0.33 as 100** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 client true** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 local-ip 10.0.0.11** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.33 next-hop 10.0.0.11** [edit]
R1	Configure iBGP peering with R4.	vyatta@R1# **set protocols bgp peer 10.0.0.44 as 100** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.44 client true** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.44 local-ip 10.0.0.11** [edit] vyatta@R1# **set protocols bgp peer 10.0.0.44 next-hop 10.0.0.11** [edit]
R1	Commit the configuration.	vyatta@R1# **commit** OK [edit]

Example 10-31 Creating route reflectors

R2	Delete confederation configuration and set local AS back to 100.	vyatta@R2# **delete protocols bgp peer 10.0.0.33** [edit] vyatta@R2# **delete protocols bgp peer 10.0.0.44** [edit] vyatta@R2# **delete protocols bgp confederation** [edit] vyatta@R2# **set protocols bgp local-as 100** [edit]
R2	Configure iBGP peering with R1.	vyatta@R2# **set protocols bgp peer 10.0.0.11 as 100** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.11 local-ip 10.0.0.22** [edit] vyatta@R2# **set protocols bgp peer 10.0.0.11 next-hop 10.0.0.22** [edit]
R2	Commit the configuration.	vyatta@R2# **commit** OK [edit]
R3	Delete confederation configuration and set local AS back to 100.	vyatta@R3# **delete protocols bgp peer 10.0.0.11** [edit] vyatta@R3# **delete protocols bgp peer 10.0.0.22** [edit] vyatta@R3# **delete protocols bgp confederation** [edit] vyatta@R3# **set protocols bgp local-as 100** [edit]
R3	Configure iBGP peering with R1.	vyatta@R3# **set protocols bgp peer 10.0.0.11 as 100** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.11 local-ip 10.0.0.33** [edit] vyatta@R3# **set protocols bgp peer 10.0.0.11 next-hop 10.0.0.33** [edit]
R3	Commit the configuration.	vyatta@R3# **commit** OK [edit]

Example 10-31 Creating route reflectors

R4	Delete confederation configuration and set local AS back to 100.	vyatta@R4# **delete protocols bgp peer 10.0.0.22** [edit] vyatta@R4# **delete protocols bgp confederation** [edit] vyatta@R4# **set protocols bgp local-as 100** [edit]
R4	Configure iBGP peering with R1.	vyatta@R4# **set protocols bgp peer 10.0.0.11 as 100** [edit] vyatta@R4# **set protocols bgp peer 10.0.0.11 local-ip 10.0.0.44** [edit] vyatta@R4# **set protocols bgp peer 10.0.0.11 next-hop 10.0.0.44** [edit]
R4	Commit the configuration.	vyatta@R4# **commit** OK [edit]

Verifying the Route Reflector

The following commands can be used to verify the route reflector configuration.

R1: show bgp peers

Example 10-32 shows the output of the **show bgp peers** command for router R1 at this stage of the configuration.

Example 10-32 Verifying route reflector on R1: “show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------      -------  --------  ---------
10.0.0.22        100    ESTAB     4    189         206      0         7
10.0.0.33        100    ESTAB     4    247         172      0         13
```

```
10.0.0.44         100     ESTAB     4     166         186       3          6
88.88.88.2        200     ESTAB     4     2569        2594      5          10

vyatta@R1>
```

R1: show bgp neighbor-routes

Example 10-33 shows the output of the **show bgp neighbor-routes** command for router R1 at this stage of the configuration.

Example 10-33 Verifying route reflector on R1: “show bgp neighbor-routes”

```
vyatta@R1> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                  Nexthop                   Peer          AS Path
------                  -------                   ----          -------
*> 172.16.0.0/19        10.0.0.11                 0.0.0.0       i
*> 2.0.0.0/24           99.99.99.2                10.0.0.44     300 i
*> 2.1.0.0/24           99.99.99.2                10.0.0.44     300 i
*> 2.2.0.0/24           99.99.99.2                10.0.0.44     300 i
*> 2.0.0.0/8            99.99.99.2                10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2               88.88.88.2     200 i
vyatta@R1>
```

R2: show bgp peers

Example 10-34 shows the output of the **show bgp peers** command for router R2 at this stage of the configuration.

Example 10-34 Verifying route reflector on R2: “show bgp peers”

```
vyatta@R2> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------      -------  --------  ---------
10.0.0.11        100    ESTAB     4    206         194      7         0

vyatta@R2>
```

R2: show bgp neighbor-routes

Example 10-35 shows the output of the **show bgp neighbor-routes** command for router R2 at this stage of the configuration.

Example 10-35 Verifying route reflector on R2: “show bgp neighbor-routes”

```
vyatta@R2> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                Nexthop                    Peer          AS Path
------                -------                    ----          -------
*> 172.16.0.0/19        10.0.0.11                  10.0.0.11       i
*> 2.0.0.0/24           99.99.99.2                 10.0.0.11       300 i
*> 2.1.0.0/24           99.99.99.2                 10.0.0.11       300 i
*> 2.2.0.0/24           99.99.99.2                 10.0.0.11       300 i
*> 2.0.0.0/8            99.99.99.2                 10.0.0.11       300 i
*> 12.0.0.0/8           88.88.88.2                 10.0.0.11       200 i
vyatta@R2>
```

As you can see, router R2 is learning all its BGP routes from peer 10.0.0.11 (R1). For example, 2.0.0.0/8 is being announced by AS 300 to router R4, router R4 is announcing it to router R1 (its RR server), and R1 is announcing it to R2 (one of its RR clients).

R3: show bgp peers

Example 10-36 shows the output of the **show bgp peers** command for router R3 at this stage of the configuration.

Example 10-36 Verifying route reflector on R3: “show bgp peers”

```
vyatta@R3> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------      -------  --------  ---------
10.0.0.11        100    ESTAB     4    189         275      13         0

vyatta@R3>
```

R3: show bgp neighbor-routes

Example 10-37 shows the output of the **show bgp neighbor-routes** command for router R3 at this stage of the configuration.

Example 10-37 Verifying route reflector on R3: “show bgp neighbor-routes”

```
vyatta@R3> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                  Nexthop                 Peer          AS Path
------                  -------                 ----          -------
*> 172.16.0.0/19        10.0.0.11               10.0.0.11     i
*> 2.0.0.0/24           99.99.99.2              10.0.0.11     300 i
*> 2.1.0.0/24           99.99.99.2              10.0.0.11     300 i
*> 2.2.0.0/24           99.99.99.2              10.0.0.11     300 i
*> 2.0.0.0/8            99.99.99.2              10.0.0.11     300 i
*> 12.0.0.0/8           88.88.88.2              10.0.0.11     200 i
vyatta@R3>
```

R4: show bgp peers

Example 10-38 shows the output of the **show bgp peers** command for router R4 at this stage of the configuration.

Example 10-38 Verifying route reflector on R4: “show bgp peers”

```
vyatta@R4> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.11        100    ESTAB     4    189        275      6         3
99.99.99.2       300    ESTAB     4    195        189      7         3

vyatta@R4>
```

R4: show bgp neighbor-routes

Example 10-39 shows the output of the **show bgp neighbor-routes** command for router R4 at this stage of the configuration.

Example 10-39 Verifying route reflector on R4: “show bgp neighbor-routes”

```
vyatta@R4> show bgp neighbor-routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                  Nexthop                 Peer          AS Path
------                  -------                 ----          -------
*> 172.16.0.0/19        10.0.0.44               0.0.0.0       i
*> 2.0.0.0/24           99.99.99.2              99.99.99.99   300 i
*> 2.1.0.0/24           99.99.99.2              99.99.99.99   300 i
*> 2.2.0.0/24           99.99.99.2              99.99.99.99   300 i
*> 2.0.0.0/8            99.99.99.2              99.99.99.99   300 i
*> 12.0.0.0/8           88.88.88.2              10.0.0.11     200 i
vyatta@R4>
```

Route Redirection

Route redirection in BGP is performed by means of routing policies. For more information about routing policies, please see “Chapter 6: Routing Policies.”

Monitoring BGP

This section presents the following topics:

- BGP Operational Commands
- Sending BGP Messages to Syslog

BGP Operational Commands

This section presents the following topics:

- Showing Best Paths in the BGP Table
- Showing BGP routes in the RIB
- Showing BGP Peers

You can use the following operational commands to monitor BGP. Please see the *Vyatta OFR Command Reference* for details on these commands and their options.

Command	Description
`show bgp dampened-routes`	Displays information about dampened BGP routes.
`show bgp neighbor-routes`	Displays the full BGP routing table.
`show bgp peers`	Displays information about BGP peerings.
`show bgp routes`	Displays best BGP paths.
`show route bgp`	Displays BGP routes stored in the routing table.
`show route ebgp`	Displays eBGP routes stored in the routing table.
`show route ibgp`	Displays iBGP routes stored in the routing table.

This section includes the following examples:

- Example 10-40 Viewing best paths in the BGP table: “show bgp routes”
- Example 10-41 Viewing BGP routes in the RIB: “show route protocol bgp”
- Example 10-42 Viewing iBGP routes: “show route protocol ibgp”

- Example 10-43 Viewing eBGP routes: “show route protocol ebgp”
- Example 10-44 Viewing BGP peer information: “show bgp peers”
- Example 10-45 Showing BGP peer information: “show bgp peers detail”

NOTE *The output in these examples may show information unrelated to the sample configurations.*

Showing Best Paths in the BGP Table

To view the best paths in the BGP table, use the **show bgp routes** command.

Example 10-40 Viewing best paths in the BGP table: “show bgp routes”

```
vyatta@R1> show bgp routes
Status Codes: * valid route, > best route
Origin Codes: i IGP, e EGP, ? incomplete

Prefix                  Nexthop                 Peer          AS Path
------                  -------                 ----          -------
*> 172.16.0.0/19        10.0.0.11               0.0.0.0       i
*> 2.0.0.0/24           99.99.99.2              10.0.0.44     300 i
*> 2.1.0.0/24           99.99.99.2              10.0.0.44     300 i
*> 2.2.0.0/24           99.99.99.2              10.0.0.44     300 i
*> 2.0.0.0/8            99.99.99.2              10.0.0.44     300 i
*> 12.0.0.0/8           88.88.88.2             88.88.88.2     200 i
vyatta@R1>
```

Showing BGP routes in the RIB

To display the routes from BGP that have been selected for inclusion in the RIB, use the **show route protocol bgp** command, as shown in Example 10-41.

Example 10-41 Viewing BGP routes in the RIB: “show route protocol bgp”

```
vyatta@R1> show route protocol bgp
Routes: 12/20, Paths: 12/20
2.0.0.0/8          [ebgp(0)]       > to 88.88.88.2       via eth0
2.0.0.0/24         [ebgp(0)]       > to 88.88.88.2       via eth0
2.1.0.0/24         [ebgp(0)]       > to 88.88.88.2       via eth0
2.2.0.0/24         [ebgp(0)]       > to 88.88.88.2       via eth0
3.0.0.0/8          [ibgp(0)]       > to 172.16.0.2       via eth1
3.0.0.0/24         [ibgp(0)]       > to 172.16.0.2       via eth1
```

```
3.1.0.0/24          [ibgp(0)]       > to 172.16.0.2       via eth1
3.2.0.0/24          [ibgp(0)]       > to 172.16.0.2       via eth1
10.1.0.0/16         [ibgp(0)]       > to 172.16.0.2       via eth1
12.0.0.0/8          [ebgp(0)]       > to 88.88.88.2       via eth0
172.16.128.0/18     [ibgp(0)]       > to 172.16.0.2       via eth1
192.168.2.0/24      [ibgp(0)]       > to 172.16.0.2       via eth1

vyatta@R1>
```

To show just iBGP routes, use the **show route protocol ibgp** option, as shown in Example 10-42.

Example 10-42 Viewing iBGP routes: “show route protocol ibgp”

```
vyatta@R1> show route protocol ibgp
Routes: 7/20, Paths: 7/20
3.0.0.0/8           [ibgp(0)]       > to 172.16.0.2       via eth1
3.0.0.0/24          [ibgp(0)]       > to 172.16.0.2       via eth1
3.1.0.0/24          [ibgp(0)]       > to 172.16.0.2       via eth1
3.2.0.0/24          [ibgp(0)]       > to 172.16.0.2       via eth1
10.1.0.0/16         [ibgp(0)]       > to 172.16.0.2       via eth1
172.16.128.0/18     [ibgp(0)]       > to 172.16.0.2       via eth1
192.168.2.0/24      [ibgp(0)]       > to 172.16.0.2       via eth1

vyatta@R1>
```

To show just eBGP routes, use the **show route protocol ebgp** option, as shown in Example 10-43.

Example 10-43 Viewing eBGP routes: “show route protocol ebgp”

```
vyatta@R1> show route protocol ibgp
Routes: 5/20, Paths: 5/20
2.0.0.0/8           [ebgp(0)]       > to 88.88.88.2       via eth0
2.0.0.0/24          [ebgp(0)]       > to 88.88.88.2       via eth0
2.1.0.0/24          [ebgp(0)]       > to 88.88.88.2       via eth0
2.2.0.0/24          [ebgp(0)]       > to 88.88.88.2       via eth0
12.0.0.0/8          [ebgp(0)]       > to 88.88.88.2       via eth0

vyatta@R1>
```

Showing BGP Peers

To view information about BGP peers, use the **show bgp peers** command in operational mode. When used with no option, this command provides a summary status for all configured BGP peers, as shown in Example 10-44.

Example 10-44 Viewing BGP peer information: “show bgp peers”

```
vyatta@R1> show bgp peers

Neighbor         AS     State     Ver  Msg Rx      Msg Tx  Update Rx  Update Tx
--------         --     -----     ---  ------     -------  --------  ---------
10.0.0.22        100    ESTAB     4    22         38       0         10
10.0.0.33        100    ESTAB     4    26         25       0         10
10.0.0.44        100    ESTAB     4    16         15       0         10
88.88.88.2       100    ESTAB     4    6          2        1         1

vyatta@R1>
```

When used with the **detail** option, this command provides detail about each peer, as shown in Example 10-45.

Example 10-45 Showing BGP peer information: “show bgp peers detail”

```
vyatta@R1> show bgp peers detail

Peer 1: local 10.0.0.11/179 remote 10.0.0.22/179
  Peer ID: 10.0.0.22
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 100
  Updates Received: 0,  Updates Sent: 6
  Messages Received: 15,  Messages Sent: 21
  Time since last received update: n/a
  Number of transitions to ESTABLISHED: 1
  Time since last in ESTABLISHED state: 355 seconds
  Retry Interval: 120 seconds
  Hold Time: 90 seconds,  Keep Alive Time: 30 seconds
 Configured Hold Time: 90 seconds,  Configured Keep Alive Time:
30 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds

Peer 2: local 10.0.0.11/179 remote 10.0.0.33/179
  Peer ID: 10.0.0.33
```

```
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 100
  Updates Received: 0,  Updates Sent: 6
  Messages Received: 13,  Messages Sent: 18
  Time since last received update: n/a
  Number of transitions to ESTABLISHED: 1
  Time since last in ESTABLISHED state: 297 seconds
  Retry Interval: 120 seconds
  Hold Time: 90 seconds,  Keep Alive Time: 30 seconds
  Configured Hold Time: 90 seconds,  Configured Keep Alive Time:
30 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds

Peer 3: local 10.0.0.11/179 remote 10.0.0.44/179
  Peer ID: 10.0.0.44
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 100
  Updates Received: 8,  Updates Sent: 6
  Messages Received: 22,  Messages Sent: 20
  Time since last received update: 338 seconds
  Number of transitions to ESTABLISHED: 1
  Time since last in ESTABLISHED state: 338 seconds
  Retry Interval: 120 seconds
  Hold Time: 90 seconds,  Keep Alive Time: 30 seconds
  Configured Hold Time: 90 seconds,  Configured Keep Alive Time:
30 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds

Peer 4: local 88.88.88.1/179 remote 88.88.88.2/179
  Peer ID: 88.88.88.88
  Peer State: ESTABLISHED
  Admin State: START
  Negotiated BGP Version: 4
  Peer AS Number: 200
  Updates Received: 5,  Updates Sent: 8
  Messages Received: 22,  Messages Sent: 25
  Time since last received update: 402 seconds
  Number of transitions to ESTABLISHED: 1
  Time since last in ESTABLISHED state: 402 seconds
  Retry Interval: 120 seconds
  Hold Time: 90 seconds,  Keep Alive Time: 30 seconds
```

```
  Configured Hold Time: 90 seconds,  Configured Keep Alive Time:
30 seconds
  Minimum AS Origination Interval: 0 seconds
  Minimum Route Advertisement Interval: 0 seconds
```

Sending BGP Messages to Syslog

The BGP process generates log messages during operation. You can configure the system to send BGP-specific log messages to syslog, by creating and enabling the **traceoptions** configuration node. The result will depend on how the system syslog is configured.

For information about configuring syslog, please refer to “Chapter 16: Logging.”

Example 10-46 enables logging for BGP.

To enable logging for BGP, perform the following steps in configuration mode:

Example 10-46 Enabling logging for BGP

Step	Command

Enable logging within the **bgp** configuration node.	`vyatta@R1# set protocols bgp traceoptions flag all disable false`
	`OK`
	`[edit]`

Commit and view the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`
	`vyatta@R1# show -all protocols bgp traceoptions`
	`  flag {`
	`  all {`
	`  disable: false`
	`  }`
	`  }`
	
	`[edit]`

Example 10-46 disables logging for BGP.

To disable logging for BGP without discarding logging configuration, perform the following steps in configuration mode:

Example 10-47 Disabling logging for BGP

Step	Command
Disable logging within the **bgp** configuration node.	vyatta@R1# **set protocols bgp traceoptions flag all disable true** OK [edit]

Commit and view the change.	vyatta@R1# **commit**
	OK
	[edit]
	vyatta@R1# **show -all protocols bgp traceoptions**
	flag {
	all {
	disable: true
	}
	}
	
	[edit]

Chapter 11: VRRP

This chapter describes how to configure the Virtual Router Redundancy Protocol on the Vyatta system.

The following topics are covered:

- VRRP Overview
- Configuring VRRP
- Viewing VRRP Information

VRRP Overview

Virtual Router Redundancy Protocol (VRRP) is a protocol for allowing a cluster of routers to act as one virtual router. VRRP, as specified by RFC 2338 and RFC 3678, was designed to provide router failover services in the event of an interface failure.

Routers in a VRRP cluster share a virtual IP address (the VIP) and a virtual MAC address. This provides alternate paths through the network for hosts without explicitly configuring them, and creates redundancy that eliminates any individual router as a single point of failure in the network. This is particularly important for statically configured default routers, the failure of which can otherwise be a catastrophic event on a network.

In VRRP, the IP addresses of interfaces on different real routers are mapped onto a “virtual router”. The virtual router is an abstract object, managed by the VRRP process, that is defined by its virtual router ID (the group identifier of the set of routers forming the virtual router) plus the virtual IP address (the “virtual address”, or VIP) presented to the network. Hosts on the network are configured to direct packets to the VIP, rather than to the IP addresses of the real interfaces.

The virtual router uses the group identifier to construct a virtual MAC address from a standard MAC prefix (specified in the VRRP standard) plus the group identifier. ARP requests for the VIP are resolved to the virtual MAC address, which “floats” from real router to real router, depending on which is acting as the master router of the virtual router. If the master router fails, the backup router is brought into service using the virtual MAC address and VIP of the virtual router. In this way, service can continue around a failed gateway transparently to hosts on the LAN.

VRRP dynamically elects the router that is to be the master. In most cases, this is simply the router with the interface having highest configured priority. If two interfaces have identical priorities, the router with the one having the highest IP address is elected master.

The master router forwards packets for local hosts and responds to ARP requests, ICMP pings, and IP datagrams directed to the VIP. Backup routers remain idle, even if healthy. ARP requests, pings, and datagrams made to the real IP addresses of interfaces are responded to by the interface in the normal way.

The master router also assumes responsibility for the virtual router, continuously forwarding advertisement packets (MAC-level multicast packets) as a “heartbeat” through TCP port 112 (the IANA well-known port for VRRP) to the backup routers. If the heartbeat stops for a configured period (the “dead interval”), VRRP assumes that the master router has become unavailable, and it elects one of the backup routers to become the new master router.

On the Vyatta system, VRRP can be run on either a standard Ethernet interface, or it can be run on the vif of an Ethernet interface (that is, a VLAN interface).

Configuring VRRP

This sequence sets up a basic VRRP configuration between two Vyatta routers.

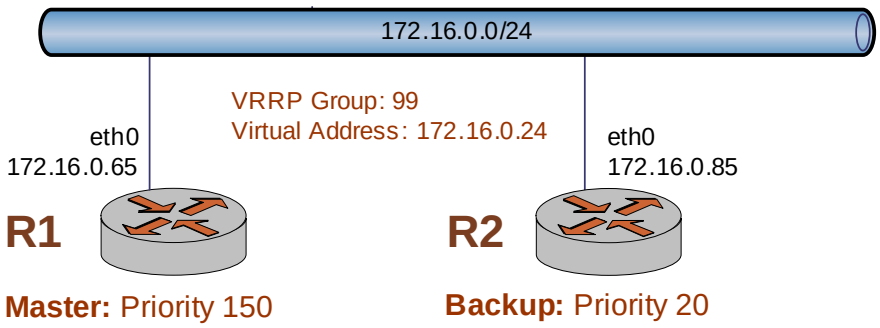
Remember that in VRRP:

- The router configured with the highest priority will initially be elected the master router. If more than one router has the highest priority, then the first active router will be elected the master router.
- Enabling preemption will allow a higher-priority neighbor to preempt the current master and become master itself.

The implementation is currently restricted to one VRRP group per interface, regardless of whether the group is defined at the physical interface level or the vif level.

In this section, sample configurations are presented for VRRP. When you have finished, the router will be configured as shown in Figure 11-1.

Figure 11-1 VRRP



This section includes the following examples:

- Example 11-1 Configuring a first router for VRRP
- Example 11-2 Configuring a backup router for VRRP

Configuring the First Router

Example 11-1 enables VRRP on eth0 of the first router (R1) and assigns it to VRRP group 99. The virtual address is 172.16.0.24. Preemption is enabled, and R1 is assigned a priority of 150.

To configure the first router for VRRP, perform the following steps in configuration mode:

Example 11-1 Configuring a first router for VRRP

Step	Command
Create the VRRP configuration node for eth0 on R1. This enables VRRP on that interface. Assign the VRRP group.	`vyatta@R1# set interfaces ethernet eth0 vrrp vrrp-group 99` `[edit]`
Specify the virtual address of the VRRP group.	`vyatta@R1# set interfaces ethernet eth0 vrrp virtual-address 172.16.0.24` `[edit]`
Enable preemption.	`vyatta@R1# set interfaces ethernet eth0 vrrp preempt true` `[edit]`
Set the priority of this router to 150.	`vyatta@R1# set interfaces ethernet eth0 vrrp priority 150` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Configuring the Second Router

Example 11-2 enables VRRP on eth0 of the second router (R1Backup), and assigns it to VRRP group 99. The virtual address is the same as that for R1: 172.16.0.24. Preemption is enabled, and R1Backup is assigned a priority of 20. This is lower than the priority of R1, so R1 will be the master and R2 will be the backup under ordinary circumstances.

To configure the second router for VRRP, perform the following steps in configuration mode:

Example 11-2 Configuring a backup router for VRRP

Step	Command
Create the VRRP configuration node for eth0 of R2. This enables VRRP on that interface. Assign the VRRP group.	`vyatta@R2# set interfaces ethernet eth0 vrrp vrrp-group 99` `[edit]`
Specify the virtual address of the VRRP group.	`vyatta@R2# set interfaces ethernet eth0 vrrp virtual-address 172.160.0.24` `[edit]`
Enable preemption.	`vyatta@R2# set interfaces ethernet eth0 vrrp preempt true` `[edit]`
Set the priority of this router to 20. This is a lower priority than that set for R1, so R1 will become the master.	`vyatta@R2# set interfaces ethernet eth0 vrrp priority 20` `[edit]`

Commit the configuration.	`vyatta@R2# commit`
	`OK`
	`[edit]`

Viewing VRRP Information

This section includes the following examples:

- Example 11-3 Showing VRRP group information
- Example 11-4 Viewing the “vrrp” configuration node for an interface

Showing VRRP Configuration

To view information about how VRRP groups are configured on the router, use the **show vrrp** command in operational mode. Example 11-3 shows VRRP configuration for R1.

Example 11-3 Showing VRRP group information

```
vyatta@R1> show vrrp
Physical interface: eth0, Address: 172.16.0.24
Interface state: up, Group: 99, State: master
Priority: 150, Advertisement interval: 1s, Authentication
   type: none
  Preempt: yes, VIP count: 1, VIP: 172.16.0.24
  Advertisement timer: 429s, Master router: 172.16.0.65
  Virtual MAC: 00:00:5E:00:01:63
```

Showing the VRRP Configuration Node

You can always view the information in configuration nodes by using the **show** command in configuration mode. VRRP is configured as a property of an interface or a vif, so in this case, you can view VRRP configuration by using the **show interfaces ethernet** *int-name* **vrrp** command, or the **show interfaces ethernet** *int-name***.***vif-name* **vrrp** command, as required. Example 11-4 shows the command you would enter to view the VRRP configuration node for interface eth0 of router R1.

To show VRRP information for an interface, perform the following step in configuration mode:

Example 11-4 Viewing the “vrrp” configuration node for an interface

Step	Command

Show the contents of the **vrrp** configuration node for eth0.	`vyatta@R1# show interfaces ethernet eth0 vrrp`
	`  vrrp-group: 99`
	`  virtual-address: 172.16.0.24`
	`  priority: 150`
	`  preempt: true`

Chapter 12: NAT

This chapter describes how to configure NAT on the Vyatta system.

The following topics are covered:

- NAT Overview
- Configuring NAT
- Viewing NAT Information

NAT Overview

Network Address Translation (NAT) is a process whereby internal addresses on a private network are mapped, or “translated” to a globally unique public IP address, or to another private address, by an intermediate device or application, such as a router, firewall, or VPN. NAT was originally designed to help conserve the number of IP addresses used by the growing number of devices accessing the Internet, but it also has important applications in network security.

The computers on the internal network can use any of the addresses set aside by the Internet Assigned Numbers Authority (IANA) for private addressing (see also RFC 1918). These are the following:

- 10.0.0.0 to 10.255.255.255 (CIDR: 10.0.0.0/8)
- 172.16.0.0 to 172.31.255.255 (CIDR: 172.16.0.0/12)
- 192.168.0.0 to 192.168.255.255 (CIDR: 192.268.0.0/16)

These reserved IP addresses are not in use on the Internet, so an external machine cannot directly route to them.

A NAT-enabled router hides the IP addresses of your internal network from the external network, by replacing the internal, private IP addresses with the public IP addresses that have been provided to it. These public IP addresses are the only addresses that are ever exposed to the external network.

The router can manage multiple public IP addresses, from which it can dynamically choose when performing address replacement (“dynamic NAT”). You should be aware that, although NAT can minimize the possibility that your internal computers make unsafe connections to the external network, it provides no protection to a computer that, for one reason or another, connects to an untrusted machine.

Therefore, you should always combine NAT with packet filtering and other features of a complete security policy to fully protect your network.

Configuring NAT

This sequence sets up a basic NAT configuration on router R1.

In this release, you must create explicit NAT rules for each direction of traffic. For example, if you configure a one-to-one static source NAT rule and you want inbound traffic to match the NAT rule, you must explicitly create a matching destination NAT rule.

Source rules egress from the trusted to the untrusted network. For static and dynamic source NAT rules, the outside address defines the IP address that faces the untrusted network. This is the address that will be substituted in for the original source IP address in packets egressing to the untrusted network.

The “source” and “destination” attributes are relative to the interface they are applied to. For example, an outbound interface will process traffic as it leaves the interface. If the type of its rule is “source,” it will change the source IP address.

An outside address is not required for source rules with a translation type of **masquerade**, because for masquerade source rules the original source IP address is replaced with the IP address of the outbound interface. In fact, if you configure a source NAT rule with a translation type of masquerade, you cannot define the outside IP address, because the system uses the primary address of the outbound interface. If you want to use one of the other IP addresses you have assigned to the interface, change the type from **masquerade** to **dynamic**. Then you will be able to define an outside address.

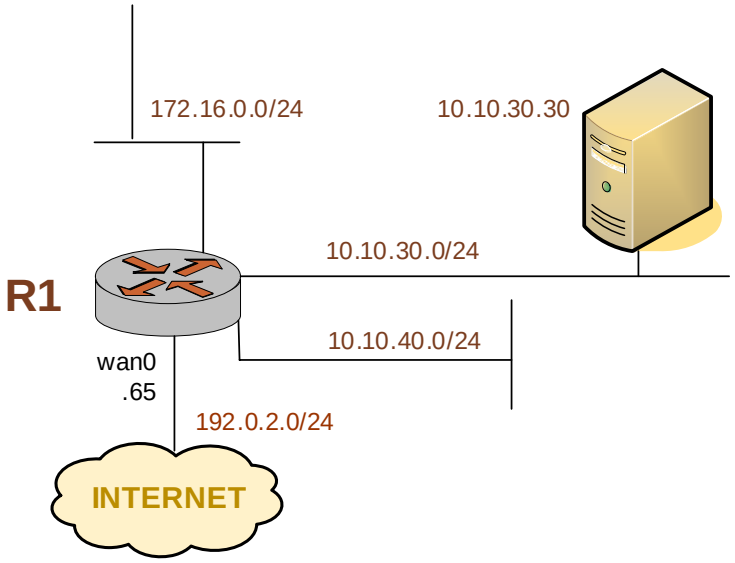
The NAT configuration structure does not currently support port rewriting (for example, where packets destined for port 80 are rewritten to be destined for 8080).

The first three rules of this configuration allow outbound traffic from three private subnets: 172.16.0.0/24, 10.10.30.0/24, and 10.10.40.0/24. Traffic from these subnets exits to the Internet through interface wan0 on R1.

The last rule of the configuration allows inbound SSH traffic to pass through wan0 on R1 to the target internal host for SSH, which is 10.10.30.30.

The subnets involved are shown in Figure 12-1.

Figure 12-1 Basic NAT configuration



First, create the three rules that allow traffic from selected internal networks to exit to the Internet through interface wan0 on R1. To create these rules, perform the following steps in configuration mode:

Example 12-1 Allowing outbound traffic from internal subnets

Step	Command
Create Rule 1, which allows outbound traffic from network 172.16.0.0/24.	vyatta@R1# **create service nat rule 1** [edit] vyatta@R1# **edit service nat rule 1** [edit service nat rule 1] vyatta@R1# **set type source** [edit service nat rule 1] vyatta@R1# **set translation-type masquerade** [edit service nat rule 1] vyatta@R1# **set outbound-interface wan0** [edit service nat rule 1] vyatta@R1# **set protocols all** [edit service nat rule 1] vyatta@R1# **set source network 172.16.0.0/24** [edit service nat rule 1] vyatta@R1# **set destination network 0.0.0.0/0** [edit service nat rule 1] vyatta@R1# **top** [edit] vyatta@R1#

Example 12-1 Allowing outbound traffic from internal subnets

Create Rule 2, which allows outbound traffic from network 10.10.30.0/24.	vyatta@R1# **create service nat rule 2** [edit] vyatta@R1# **edit service nat rule 2** [edit service nat rule 2] vyatta@R1# **set type source** [edit service nat rule 2] vyatta@R1# **set translation-type masquerade** [edit service nat rule 2] vyatta@R1# **set outbound-interface wan0** [edit service nat rule 2] vyatta@R1# **set protocols all** [edit service nat rule 2] vyatta@R1# **set source network 10.10.30.0/24** [edit service nat rule 2] vyatta@R1# **set destination network 0.0.0.0/0** [edit service nat rule 2] vyatta@R1# **top** [edit] vyatta@R1#
Create Rule 3, which allows outbound traffic from network 10.10.40.0/24.	vyatta@R1# **create service nat rule 3** [edit] vyatta@R1# **edit service nat rule 3** [edit service nat rule 3] vyatta@R1# **set type source** [edit service nat rule 3] vyatta@R1# **set translation-type masquerade** [edit service nat rule 3] vyatta@R1# **set outbound-interface wan0** [edit service nat rule 3] vyatta@R1# **set protocols all** [edit service nat rule 3] vyatta@R1# **set source network 10.10.40.0/24** [edit service nat rule 3] vyatta@R1# **set destination network 0.0.0.0/0** [edit service nat rule 3] vyatta@R1# **top** [edit] vyatta@R1#
Commit the configuration.	vyatta@R1# **commit** OK [edit]

Tip: *Where public IP addresses would normally be used, the example uses RFC 3330 "TEST-NET" IP addresses (192.0.2.0/24)*

Now, create the rule that accepts SSH traffic directed at the public IP address of wan0 on R1 (192.0.2.65) to pass through to a single internal host at the private IP address 10.10.30.30.

Note that, in effect, this configuration "exports" the private server "outside" the protected network. This means that you will not be able to access the router from outside using SSH. That is, trying to access address 192.0.2.65 will now access the SSH server rather than the Vyatta system.

To create this rule, perform the following steps in configuration mode:

Example 12-2 Allowing inbound SSH traffic

Step	Command
Create Rule 4, which allows inbound SSH traffic destined for the public IP address.	vyatta@R1# **create service nat rule 4** [edit] vyatta@R1# **edit service nat rule 4** [edit service nat rule 4] vyatta@R1# **set type destination** [edit service nat rule 4] vyatta@R1# **set translation-type stati**c [edit service nat rule 4] vyatta@R1# **set inbound-interface wan0** [edit service nat rule 4] vyatta@R1# **set protocols tcp** [edit service nat rule 4] vyatta@R1# **set source network 0.0.0.0/0** [edit service nat rule 4] vyatta@R1# **set destination address 192.0.2.65** [edit service nat rule 4] vyatta@R1# **set destination port-name ssh** [edit service nat rule 4] vyatta@R1# **set inside-address address 10.10.30.30** [edit service nat rule 4] vyatta@R1# **top** [edit service nat rule 4] vyatta@R1#
Commit the configuration.	vyatta@R1# **commit** OK [edit]

Viewing NAT Information

This section includes the following examples:

- Example 12-3 Showing NAT rules
- Example 12-4 Viewing the “service nat” configuration node

Showing NAT Rules

To display NAT rules, use the **show nat rules** command in operational mode, as shown in Example 12-3.

Example 12-3 Showing NAT rules

```
vyatta@R1> show nat rules
```

Showing NAT Configuration

You can always view the information in configuration nodes by using the **show** command in configuration mode. NAT is a protocol service, so in this case you can view NAT configuration by using the **show service nat** command, as shown in Example 12-4.

To show the information in the **service nat** configuration node, perform the following step in configuration mode:

Example 12-4 Viewing the “service nat” configuration node

Step	Command

Show the contents of the **service nat** configuration node.	`vyatta@R1# show service nat`
	`  rule 1 {`
	`  type: "source"`
	`  translation-type: "masquerade"`
	`  inbound-interface: "wan0"`
	`  protocols: "all"`

Example 12-4 Viewing the “service nat” configuration node

```
        source {
            network: 172.16.0.0/24
        }
        destination {
            network: 0.0.0.0/0
        }
    }
    rule 2 {
        type: "source"
        translation-type: "masquerade"
        inbound-interface: "wan0"
        protocols: "all"
        source {
            network: 10.10.30.0/24
        }
        destination {
            network: 0.0.0.0/0
        }
    }
    rule 3 {
        type: "source"
        translation-type: "masquerade"
        inbound-interface: "wan0"
        protocols: "all"
        source {
            network: 10.10.30.0/24
        }
        destination {
            network: 0.0.0.0/0
        }
    }
```

Example 12-4 Viewing the “service nat” configuration node

```
                rule 4 {
                    type: "destination"
                    translation-type: "static"
                    inbound-interface: "wan0"
                    protocols: "tcp"
                    source {
                        network: 0.0.0.0/0
                    }
                    destination {
                        address: 99.99.99.65
                        port-name: “ssh”
                    }
                    inside-address {
                        address: 10.10.30.30
                    }
                }
            [edit]
            vyatta@R1#
```

Chapter 13: Firewall

This chapter describes how to configure the Firewall on the Vyatta system.

The following topics are covered:

- Firewall Overview
- Configuring the Firewall
- Viewing Firewall Information

Firewall Overview

The Vyatta system's firewall functionality analyzes and filters IP packets between network interfaces. The most common application of this is to protect traffic between an internal network and the Internet. It allows you to filter packets based on their characteristics and perform actions on packets that match the rule. It provides:

- Packet filtering can be performed for traffic traversing the router, using "in" and "out" on an interface. Packets destined to the router itself can be filtered using the "local" keyword.
- Criteria that can be defined for packet-matching rules include source IP address, destination IP address, source port, destination port, IP protocol, and ICMP type.
- General detection on IP options such as source routing and broadcast packets

The Vyatta system The Vyatta firewall features stateful packet inspection and can provide significant additional protection in a layered security strategy. The router can intercept network activity, categorize it against its configured database of permitted traffic, and allow or deny the attempt. This adds add an extra layer of security when used in conjunction with stateful packet-filtering devices.

To use the firewall feature, you define a firewall rule set as a named firewall instance. You then apply the firewall instance to interfaces, where the instance acts as a packet filter. The firewall instance will filter packets in one of the following ways, depending on what you specify when you apply the firewall instance:

- **in.** If you apply the rule set as **in**, the firewall will filter packets entering the interface.
- **out.** If you apply the rule set as **out**, the firewall will filter packets leaving the interface.
- **local.** If you apply the rule set as **local**, the firewall will filter packets destined for the router directly connected to this interface.

For each interface, you can apply up to three firewall instances: one firewall **in** instance, one firewall **out** instance, and one firewall **local** instance.

Note that after the final user-defined rule in a rule set is executed, an implicit rule of **deny all** takes effect.

Make sure the firewall instance you apply to an interface is already defined, or you may experience unintended results. If you apply a firewall instance that does not exist to an interface, the implicit firewall rule of **allow all** will be applied.

Configuring the Firewall

This section sets up a basic firewall configuration. To configure the firewall:

1 You define a number of named firewall rule sets. This includes:

- Specifying match conditions for traffic.
- Specifying the action to be taken if traffic matches the specified criteria. Traffic can be **accepted**, silently **dropped**, or **rejected** with a TCP reset.

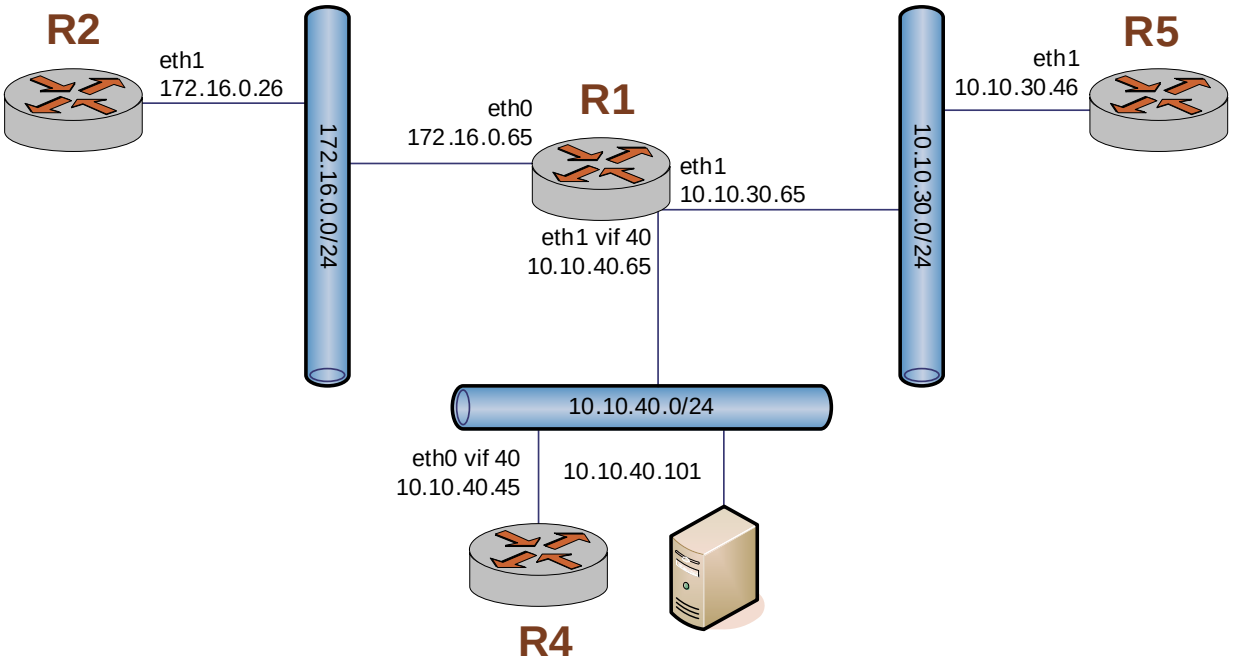
2 You apply the named rule sets to an interface as packet filters. You can apply one named rule set to each of the following:

- **in.** If you apply the rule set as **in**, the firewall will filter packets entering the interface.
- **out.** If you apply the rule set as **out**, the firewall will filter packets leaving the interface.
- **local.** If you apply the rule set as **local**, the firewall will filter packets destined for the router directly connected to this interface.

Note that after the final user-defined rule in a rule set is executed, an implicit rule of **reject all** takes effect.

This section presents a sample configuration for firewall. When you have finished, the router will be configured on router R1 as shown in Figure 13-1.

Figure 13-1 Firewall



This section includes the following examples:

- Example 13-1 Filtering on source IP
- Example 13-2 Filtering on source and destination IP
- Example 13-3 Filtering on source IP and destination protocol
- Example 13-4 Defining a network-to-network filter

Filter on Source IP

Example 13-1 defines a firewall rule set containing one rule, which filters on source IP address only. This rule will deny packets coming from router R2. It then applies the firewall rule set to packets inbound on interface eth0.

To create a rule set that filters on source IP, perform the following steps in configuration mode:

Example 13-1 Filtering on source IP

Step	Command
Create the configuration node for FWTEST-1 and its rule Rule 1. This rule rejects traffic matching the specified criteria.	`vyatta@R1# set firewall name FWTEST-1 rule 1 action reject` `[edit]`
This rule applies to traffic that has 176.16.10.26 as the source.	`vyatta@R1# set firewall name FWTEST-1 rule 1 source address 172.16.10.26` `[edit]`
Apply FWTEST-1 to inbound packets on eth0.	`vyatta@R1# set interfaces ethernet eth0 firewall in name FWTEST-1` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Filter on Source and Destination IP

Example 13-2 defines another firewall rule set. It contains one rule, which filters on both source and destination IP address. This rule accepts packets leaving R5 through eth1 using 10.10.30.46, and destined for 10.10.40.101. It then applies the firewall rule set to packets outbound from vif 1 on interface eth1.

To create a rule set that filters on source and destination IP, perform the following steps in configuration mode:

Example 13-2 Filtering on source and destination IP

Step	Command
Create the configuration node for FWTEST-2 and its rule Rule 1. This rule accepts traffic matching the specified criteria.	`vyatta@R1# set firewall name FWTEST-2 rule 1 action accept` `[edit]`

Example 13-2 Filtering on source and destination IP

This rule applies to traffic that has 10.10.30.46 as the source.	`vyatta@R1# set firewall name FWTEST-2 rule 1 source address 10.10.30.46` `[edit]`
This rule applies to traffic that has 10.10.40.101 as the destination.	`vyatta@R1# set firewall name FWTEST-2 rule 1 destination address 10.10.40.101` `[edit]`
Apply FWTEST-2 to outbound packets on eth1 vif 40.	`vyatta@R1# set interfaces ethernet eth1 vif 40 firewall out name FWTEST-2` `[edit]`
Commit the configuration.	`vyatta@R1# commit` `[edit]`

Filter on Source IP and Destination Protocol

Example 13-3 defines a firewall rule that filters on source IP address and destination protocol. This rule allows TCP packets originating from address 10.10.30.46 (that is, R5), and destined for the Telnet port of R1. The rule set is applied to local packets (that is, packets destined for this router, R1) through eth1.

To create a rule set that filters on source IP and destination protocol, perform the following steps in configuration mode:

Example 13-3 Filtering on source IP and destination protocol

Step	Command
Create the configuration node for FWTEST-3 and its rule Rule 1. This rule accepts traffic matching the specified criteria.	`vyatta@R1# set firewall name FWTEST-3 rule 1 action accept` `[edit]`
This rule applies to traffic that has 10.10.30.46 as the source.	`vyatta@R1# set firewall name FWTEST-3 rule 1 source address 10.10.30.46` `[edit]`
This rule applies to TCP traffic.	`vyatta@R1# set firewall name FWTEST-3 rule 1 protocol tcp` `[edit]`
This rule applies to traffic that is destined for the Telnet service.	`vyatta@R1# set firewall name FWTEST-3 rule 1 destination port-name telnet` `[edit]`
Apply FWTEST-3 to packets bound for this router arriving on eth1.	`vyatta@R1# set interfaces ethernet eth1 firewall local name FWTEST-3` `[edit]`

Example 13-3 Filtering on source IP and destination protocol

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Defining a Network-to-Network Filter

Example 13-4 creates a network-to-network packet filter, allowing packets originating from 10.10.40.0/24 and destined for 172.16.0.0/24. It then applies the firewall rule set to packets inbound through vif 40 on interface eth1.

To create a network-to-network filter, perform the following steps in configuration mode:

Example 13-4 Defining a network-to-network filter

Step	Command
Create the configuration node for FWTEST-4 and its rule Rule 1. This rule accepts traffic matching the specified criteria.	`vyatta@R1# set firewall name FWTEST-4 rule 1 action accept` `[edit]`
This rule applies to traffic coming from the network 10.10.40.0/24.	`vyatta@R1# set firewall name FWTEST-4 rule 1 source network 10.10.40.0/24` `[edit]`
This rule applies to traffic destined for the network 172.16.0.0/24.	`vyatta@R1# set firewall name FWTEST-4 rule 1 destination network 172.16.0.0/24` `[edit]`
Apply FWTEST-4 to packets bound for this router arriving through vif 40 on eth1.	`vyatta@R1# set interfaces ethernet eth1 vif 40 firewall in name FWTEST-4` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Viewing Firewall Information

This section includes the following examples:

- Example 13-5 Showing a firewall rule set
- Example 13-6 Showing firewall configuration on an interface
- Example 13-7 Displaying the “firewall” configuration node
- Example 13-8 Showing the firewall configuration of an interface

Showing Firewall Rule Set Information

You can see how firewall rule sets are set up by using the **show firewall** command in operational mode and specifying the name of the rule set. Example 13-5 shows what you configured for firewall rule set FWTEST-1.

Example 13-5 Showing a firewall rule set

```
vyatta@R1> show firewall FWTEST-1

State Codes: E - Established, I - Invalid, N - New, R - Related

rule  action  source            destination         proto  state
----  ------  ------            -----------         -----  -----
1     REJECT  172.16.10.26       0.0.0.0/0            all    any
1025  DROP    0.0.0.0/0          0.0.0.0/0                   any


vyatta@R1>
```

Showing Firewall Configuration on Interfaces

Example 13-6 shows how firewall rule set FWTEST-1 is applied to interface eth0.

Example 13-6 Showing firewall configuration on an interface

```
vyatta@R1# show interfaces ethernet eth0 firewall
   in {
      name: "FWTEST-1"
   }

[edit]
vyatta@R1#
```

Showing Firewall Configuration

You can always view the information in configuration nodes by using the **show** command in configuration mode. In this case you can view firewall configuration by using the **show firewall** command in configuration mode, as shown in Example 13-7.

Example 13-7 Displaying the “firewall” configuration node

```
vyatta@R1# show firewall
    name "FWTEST-1" {
        rule 1 {
            action: "reject"
            source {
                address: 172.16.10.26
            }
        }
    }
    name "FWTEST-2" {
        rule 1 {
            action: "accept"
            source {
                address: 10.10.30.46
            }
            destination {
                address: 10.10.40.101
            }
        }
    }
    name "FWTEST-3" {
        rule 1 {
```

```
            protocol: "tcp"
            action: "accept"
    source {
                address: 10.10.30.46
            }
            destination {
                port-name: "telnet"
            }
        }
    }
    name "FWTEST-4" {
        rule 1 {
            action: "accept"
            source {
                network: 10.10.40.0/24
            }
            destination {
                network: 172.16.0.0/24
            }
        }
    }

[edit]
vyatta@R1#
```

To see what firewall rule sets have been applied to a given interface, use the **show interfaces ethernet** *int-name* **firewall** command, as shown in Example 13-8.

Example 13-8 Showing the firewall configuration of an interface

```
vyatta@R1> configure
Entering configuration mode.
There are no other users in configuration mode.
vyatta@R1# show interfaces ethernet eth0 firewall
    in {
        name: "FWTEST-1"
    }

[edit]
vyatta@R1#
```

Chapter 14: IPsec VPN

This chapter describes how to configure IPsec VPN on the Vyatta router.

The following topics are covered:

- IPsec VPN Overview
- Committing VPN Configuration Changes
- Configuring a Basic Site-to-Site Connection
- Authenticating with RSA Digital Signatures
- Defining a Connection with NAT
- Configuring IPsec Tunnels between Three Gateways
- Monitoring IPsec VPN

IPsec VPN Overview

This section presents the following topics:

- IPsec Architecture
- IPsec Phase 1 and Phase 2
- IKE Key Exchange
- Encryption Ciphers
- Hash Algorithms
- Pre-Shared Keys
- Digital Signatures
- Diffie-Hellman Groups
- Main Mode
- Aggressive Mode
- Perfect Forward Secrecy

An IPsec Virtual Private Network (VPN) is a virtual network that operates across the public network, but remains “private” by establishing encrypted tunnels between two or more end points. VPNs provide:

- **Data integrity.** Data integrity ensures that no one has tampered with or modified data while it traverses the network. Data integrity is maintained with hash algorithms.
- **Authentication.** Authentication guarantees that data you receive is authentic; that is, that it originates from where it is supposed to, and not from someone masquerading as the source. Authentication is also ensured with hash algorithms.
- **Confidentiality.** Confidentiality ensures data is protected from being examined or copied while transiting the network. Confidentiality is accomplished using encryption.

An IP Security (IPsec) VPN secures communications and access to network resources for site-to-site access using encryption, authentication, and key management protocols. On a properly configured VPN, communications are secure, and the information that is passed is protected from attackers.

The Vyatta router currently supports site-to-site IPsec VPN connectivity. Site-to-site VPN connections are normally established between two (or more) VPN gateways and provide connectivity for user hosts, servers, and other devices at each location. Connectivity is normally based on IP source and destination network pairs, allowing multiple hosts to share the same tunnel between locations.

Site-to-site VPNs enable enterprises to create low-cost connectivity between offices. These site-to-site VPNs frequently replace more expensive WAN technologies such as private lines or Frame Relay.

IPsec Architecture

IPsec is a suite of protocols designed to provide end-to-end security at the network layer (Layer 3), using encryption and authentication techniques. From the point of view of IP networking equipment, encrypted packets can be routed just like any other ordinary IP packets. The only devices that require an IPsec implementation are the IPsec endpoints.

There are three main components of the IPsec architecture. These are:

- The Authentication Header (AH) protocol.
- The Encapsulating Security Payload (ESP) protocol
- The Internet Key Exchange (IKE) protocol, formerly referred to as ISAKMP/Oakley

Of these, the Vyatta router currently supports ESP, which encrypts the packet payload and prevents it from being monitored, and IKE, which provides a secure method of exchanging cryptographic keys and negotiating authentication and encryption methods.

The set of IPsec parameters describing a connection is called a *security policy.* The security policy describes how both endpoints will use security services, such as encryption, hash algorithms, and Diffie-Hellman groups, to communicate securely.

The IPsec peers negotiate a set of security parameters, which must match on both sides. Then they create a *security association* (SA). An IPsec SA describes the connection in one direction. For packets to travel in both directions in a connection, both an inbound and an outbound SA are required.

IPsec Phase 1 and Phase 2

The establishment of an IPsec connection takes place in two phases, called IKE phases:

- In IKE Phase 1, the two endpoints authenticate one another and negotiate keying material. This results in an encrypted tunnel used by Phase 2 for negotiating the ESP security associations.
- In IKE Phase 2, the two endpoints use the secure tunnel created in Phase 1 to negotiate ESP SAs. The ESP SAs are what are used to encrypt the actual user data that is passed between the two endpoints.

IKE Phase 1 establishes an ISAKMP SA (typically called an IKE SA). The IKE protocol is used to dynamically negotiate and authenticate keying material and other security parameters required to provide secure communications. IKE itself uses a combination of four protocols (including ISAKMP and Oakley) to dynamically manage keys in the context of IPsec.

If the IKE Phase 1 negotiation is successful, then the ISAKMP SA is established. The ISAKMP SA essentially contains the information from the “winning proposal” of the negotiation, recording the security encryption and keying material that was successfully

negotiated. This creates a secure “control channel” where keys and other information for protecting Phase 2 negotiation are maintained. The ISAKMP SA encrypts only Phase 2 ESP security association negotiations, plus any IKE messages between the two endpoints.

An ISAKMP SA is maintained for a pre-determined lifetime. This lifetime is configured, not negotiated or passed between peers. The configured lifetime may be different between peers. When the configured lifetime expires, a new ISAKMP SA is negotiated.

IKE Phase 2 negotiations are also managed by the IKE protocol. Using the encryption provided by the security association, the security policy is used to try and negotiate a Phase 2 SA. The security policy includes information about the communicating hosts and subnets, as well as the ESP information for providing security services for the connection, such as encryption cipher and hash algorithm. If the IKE Phase 2 negotiation process is successful, a pair of ESP SAs (typically called IPsec SAs) is established—one inbound and one outbound—between the two endpoints. This is the encrypted VPN “tunnel” between the two endpoints. At this point, the user data can be exchanged through the encrypted tunnel.

Between any two IPsec VPN peers, there can be just one control channel for exchanging Phase 2 keying material. This means that between any two peers there will be just one ISAKMP SA on each peer.

However, between two VPN peers, any number of security policies can be defined. For example, you can define a security policy that creates a tunnel between two hosts, and a different security policy that creates a tunnel between a host and a subnet, or between two subnets. Since multiple tunnels can exist between two peers, this means that multiple IPsec SAs can be active at any time between two peers.

IKE Key Exchange

To be able to create an ISAKMP SA, the two devices must agree on all of the following:

- The encryption algorithm
- The bit-strength of the encryption key (Diffie-Hellman group)
- The authentication method
- The hash algorithm
- The authentication material (pre-shared secret)

All of this information is contained in an *IKE Phase 1 proposal.* A VPN gateway can be configured multiple Phase 1 proposals. Note that the SA lifetime is not negotiated.

During an IKE key exchange, one device (the *initiator*) sends the first packet in the exchange. This first packet consist of all the Phase 1 proposals configured for this VPN peer, in a sequence. This set of proposals informs the other gateway of what security and authentication policies it supports. The second device (the *responder*) inspects the set of

proposals and returns the policy representing strongest security policy that both devices can agree on. If this process is successful, both devices agree on the parameter and the ISAKMP SA is established.

Once the ISAKMP SA has been established, the two devices can use this SA to encrypt the Phase 2 traffic where the two endpoints try to negotiate an IPsec SA for each matching security policy that has been configured between the two endpoints. Only after the IPsec SAs have been established can IPsec traffic be passed.

Different devices initiate IKE negotiation differently. Many VPN devices bring up VPN tunnels only on demand. These devices monitor traffic to see if it is “interesting”—that is, to see if it matches a configured security policy. Once the device receives traffic matching a specific security policy, the device will attempt to negotiate an IPsec SA that will be used to encrypt that traffic.

Other devices, including the Vyatta system, will attempt to initiate Phase 2 negotiations as soon as a correct policy configuration is entered. If both endpoints behave in this way, a race condition can occur, where duplicate IPsec SAs get created.

Encryption Ciphers

Ciphers are used to encrypt data, so that it cannot be read or monitored during transit. The Vyatta router supports the following encryption ciphers:

Table 14-1 Supported encryption ciphers

Cipher	Description
AES	The Advanced Encryption Standard (AES) is a U.S. government standard that was developed to take the place of DES, which has become easier to break using the more powerful computers available today. AES can run very quickly for a block cipher and can be implemented in a relatively small space. It has a block length which can vary between 192 and 256 bits, and a key length that can range between 128 and 256 bits in increments of 32 bits. The Vyatta router supports AES with a 128-bit key and with a 256-bit key.
3DES	Triple-DES is a variant of the Data Encryption Standard (DES). DES was formerly the most commonly used cipher, but in recent years has been compromised, and is no longer recommended as a first choice. The Vyatta router only supports Triple-DES. Triple-DES is an iterative block cipher, where DES is used in three consecutive iterations on the same block of text, where either two or three keys are used. The resulting ciphertext is much harder to break than DES. Using two keys yields 112 bits key strength; using 3 keys yields 168 bits key strength.

Hash Algorithms

A hash function is a cryptographic algorithm used for message authentication. A hash function takes a message of arbitrary length and produces an output of fixed length, called a message digest or fingerprint. Hash functions are used to verify that messages have not been tampered with.

The Vyatta router supports the following hash functions:

Table 14-2 Supported hash functions

Cipher	Description
MD5	MD5 is the most recent version of message digest algorithm. MD5 takes a message of arbitrary length and produces a 128-bit condensed digital representation, called a message digest. It is often used when a large file must be compressed and encrypted, then signed with a digital signature. Message digest is quite fast and efficient compared with SHA-1, because it uses primitive operations and produces a shorter message. However, it is not as secure as SHA-1, and has reportedly been compromised in some ways, though not yet in ways that make it insecure.
SHA-1	SHA stands for Secure Hash Algorithm, also known as the Secure Hash Standard. The SHA hash functions are five one-way cryptographic algorithms for computing a message digest. SHA-1 is an extension of the original SHA, and is the standard hash algorithm supported by the U.S. government. SHA-1 takes a message of arbitrary string length (the message must be smaller than 2^64 bits) and produces a 160-bit message digest. SHA-1 is slower than MD5, but it is more secure, because the additional bits in the message digest provide more protection from brute-force attacks.

Pre-Shared Keys

A pre-shared secret, or pre-shared key (PSK), is a method of authentication. The secret, or key, is a string agreed upon beforehand by both parties as key for authenticating the session. It is used to generate a hash such that each VPN endpoint can authenticate the other.

Note that the pre-shared secret, although an ordinary string, is not a “password.” It is actually used to generate a hashed key to form a “fingerprint” proving the identity of each endpoint. This means that long complex strings are more secure than short strings. Choose complex pre-shared secrets and avoid short ones, which can be more easily compromised by an attack.

The preshared secret is not passed during IKE negotiation. It is configured on both sides, and must match on both sides.

A pre-shared secret is an example of *symmetric cryptography:* the key is the same on both sides. Symmetric encryption algorithms are less computationally intensive than asymmetric algorithms, and are therefore faster. However, in symmetric cryptography, the two communicating parties must exchange keys in advance. Doing this securely can be a problem.

Pre-shared secret and digital signatures are the most common methods of IKE authentication. Pre-shared secret is an easy and effective way to quickly set up authentication with little administrative overhead. However, it has several drawbacks.

- If a pre-shared key is captured and no one is aware of it, the attacker has access to your network as long as that key is in use.
- Pre-shared secrets are manually configured, so they should be regularly changed. However, this task is often falls off the list of busy network administrators. Using pre-shared key values with remote users is equivalent to gicing them a password to your network.

NOTE *You should restrict the use of pre-shared keys to smaller, low-risk environments.*

Digital Signatures

Along with pre-shared key, RSA digital signatures are the most common means of IKE authentication.

An RSA digital signature is based on a cryptographic key that has two parts: a public part and a private part. One part (the public key) is widely shared, and may even be publicly distributed. The other part (the private key) remains secret. These keys are mathematically related but are independent, so that neither key is derivable from the other.

The key is used as input to a hash function; together, the key and the hash function form a signing function that, when applied to a document, creates a digital signature.

An RSA key can be used either to encrypt or authenticate, and this is based on two facts:

- Data encrypted with the agent's public key can only be decrypted by the agent, using the private key. This means that any peer can send information securely by encrypting it with the public key and forwarding it to the agent.
- Data processed with a hash function can be encrypted with the signer's private key—such data is said to be *digitally signed*. Since anyone with the public key can verify the digital signature, this communication can be accepted as authentically coming from the agent.

The algorithms that encrypt using RSA keys are very secure but extremely slow—so slow that it would be impracticable to encrypt an entire set of data using them. Instead, the agent produces a digital signature for the data, as follows:

1. A hash function is applied to the data to generate a message digest. The message digest is much shorter than the original data, and any peer possessing the same hash function can produce the identical message digest.
2. The private key is used to encrypt the message digest. This encrypted message digest is the digital signature.
3. The original message and the digital signature are all sent to the peer in an encrypted packet. (The encryption of the packet is independent of the digital signature.)
4. When the peer receives the packet, it decrypts the packet. Then it uses the sending agent's public key to decrypt the digital signature. This recovers the message digest.
5. The peer applies the hash function to the original message (which was also sent in the packet) and compares the resulting message digest to the message digest recovered from the digital signature.
    - If the message digests match, the peer can accept the communication as authentic.
    - If the message digests do not match, the peer must consider the communication to have been tampered with, or corrupted in some other way, and reject it.

When the system generates an RSA digital signature, it stores in a file. The file containing the digital signature contains both the public key part and the private key part of the digital signature. When you view the RSA key, by looking at VPN configuration or by using the **show vpn ike rsa-keys** command, only the public key displays (along with any public keys configured for VPN peers). It is the public key that you should share with the other VPN peer.

By default, the RSA digital signature file for the local host is stored in the file **/etc/ipsec.d/rsa-keys/localhost.key**. When the key is required to authenticate the VPN peer, this is where the system looks for it. You can change the location and name of the file through configuration.

You can only have one RSA digital signature configured for the local host. If you generate a new key, it overwrites the previous key.

Diffie-Hellman Groups

Diffie-Hellman key exchange is a cryptographic protocol for securely exchanging encryption keys over an insecure communications channel, such as the Internet. Diffie-Hellman key exchange was developed in 1976 by Whitfield Diffie and Martin Hellman. It is based on two facts:

- Asymmetric encryption algorithms are much more secure than symmetric algorithms, which require that two parties exchange secret keys in advance. However,
- Asymmetric algorithms are much slower and much more computationally expensive than symmetric algorithms.

In a Diffie-Hellman key exchange, asymmetric cryptography is used at the outset of the communication (IKE Phase 1) to establish a shared key. Once the key has been exchanged, it can then be used symmetrically to encrypt subsequent communications (IKE Phase 2).

Diffie-Hellman key exchange uses a group of standardized global unique prime numbers and generators to provide secure asymmetric key exchange. The original specification of IKE defined four of these groups, called Diffie-Hellman groups or Oakley groups. Since then, a fifth has been defined.

The Vyatta router supports the following Diffie-Hellman groups:

Table 14-3 Supported Diffie-Hellman groups

Diffie-Hellman Group	Description
2	Diffie-Hellman group 2 is a modular exponentiation group (MODP). This is a 1024-bit group with a key strength approximately equivalent to symmetric keys of length 80 bits.
5	Diffie-Hellman group 5 is a 1536-bit modular exponentiation (MODP) group. This is a 1024-bit group with a key strength approximately equivalent to symmetric keys of length 128 bits.

Main Mode

Under ordinary conditions, establishing the ISAKMP SA requires several packets to be sent and received:

- The first two messages determine communications policy.
- The next two messages exchange Diffie-Hellman public data.
- The last two messages authenticate the Diffie-Hellman exchange.

This is the normal method of establishing a successful Phase 1 connection, and it is called *main mode*. This method provides the most security and privacy, because authentication information is not exchanged until a full Diffie-Hellman exchange has been negotiated and encryption has been enabled. It also affords the most flexibility, because there are more message exchanges in which to negotiate suitable options. At the same time, main mode is slower because of the exchange of so many computationally expensive messages.

Aggressive Mode

Phase 1 negotiation can generate a fair bit of latency, and some vendors required a faster mode, in which fewer packets were needed. This mode is called *aggressive mode*. In aggressive mode, the number of round-trip messages is reduced:

- The first two messages determine a communications policy and exchange Diffie-Hellman public data. The information required for the security association, key exchange, and authentication are all transmitted at once.
- A third message authenticates the responder, and this completes the negotiation.

This means that the hash of the authentication key is exchanged before encryption has been enabled. If these packets were to be captured, they could be used in an attack against the system. Also, because the authentication material is sent in the initial packet, a Denial-of-Service attack can be accomplished just by bombarding it with initial aggressive mode packets.

In addition, because fewer messages are exchanged, there are fewer opportunities to negotiate options. That is, there can be multiple IKE proposals for Phase 1 negotiation. In in main mode, the VPN peers can “fall through” the proposals until they find one that matches, but in aggressive mode, the first proposal in the policy must match or the connection will fail. This may lead to more failures in aggressive mode to establish mutually acceptable security associations.

NOTE *You should avoid using aggressive mode, unless you are working in an environment known to be secure.*

Perfect Forward Secrecy

In Perfect Forward Secrecy (PFS), the private key is used to generate a temporary key (the session key) that is used for a short time and then discarded. Subsequent keys are independent of any previously created keys. This way, if a key is compromised, it does not affect any further keys, or compromise the security of data protected by other keys.

PFS provides a way to optimize both efficiently and security. Reasonably-sized keys are much more computationally efficient than large keys, but are also less secure. In PFS, you can use reasonably-sized keys and refresh them frequently.

Committing VPN Configuration Changes

An IPsec VPN connection includes multiple components, some of which are interdependent. For example, a VPN connection configuration requires a valid IKE group configuration, a valid ESP group configuration, and a valid tunnel configuration. In addition, the interface specified in the connection must be enabled for IPsec VPN. When you commit a VPN configuration, the Vyatta router performs a full verification on the configuration. If any required component is missing or incorrectly specified, the commit will fail.

For an IPsec VPN site-to-site connection configuration to successfully commit, all the following must be correctly configured:

- The interface and IP address must already be configured.

- The interface must be enabled for IPsec VPN.
- The peer must be configured.
- The IKE group specified in the peer configuration must be defined.
- The tunnel must be configured.
- The ESP group specified in the tunnel must be defined.
- The local IP address specified for the peer must be configured on the VPN-enabled interface.

In addition, please note that modifying global parameters (such as **ipsec-interface** or **nat-traversal**) requires an IPsec restart, and therefore restarts all tunnels.

Adding, modifying, or deleting a tunnel restarts only the modified tunnel. Modifying an existing IKE group or ESP group restarts any tunnel using the group. Changing authentication information (pre-shared key or RSA signature) does not result in a tunnel restart.

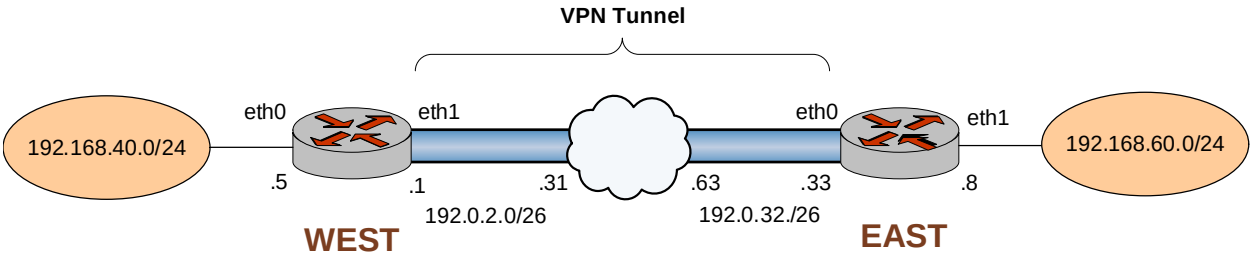
Configuring a Basic Site-to-Site Connection

This section presents the following topics:

- Configure WEST
- Configure EAST

This section presents a sample configuration for a basic IPsec tunnel between Vyatta systems WEST and EAST. First WEST is configured, and then EAST. When you have finished, these peers will be configured as shown in Figure 14-1.

Figure 14-1 Basic site-to-site IPsec VPN connection



Before you begin:

- In this set of examples, we assume that you have two Vyatta routers, with host names configured WEST and EAST. (The example routers are configured with the host name in upper case.) The last set of examples assumes you have a third Vyatta router with host name SOUTH.
- Any Ethernet interface to be used for IPsec VPN must already be configured. In this example, you will need eth1 on WEST and eth0 on EAST, plus internal subnet information.
- The interface must be configured with the IP address you want to use as the source IP for packets sent to the peer VPN gateway. In this example, IP address 192.0.2.1 is defined on eth1 of WEST, and 192.0.2.33 is defined on eth0 of EAST.

Tip: *Where public IP addresses would normally be used, the example uses RFC 3330 "TEST-NET" IP addresses (192.0.2.0/24)*

Please see "Chapter 2: Ethernet and VLAN Interfaces" for information on configuring Ethernet interfaces and IP addresses.

NOTE *The sending and receiving of ICMP redirects is disabled when IPsec VPN is configured.*

Configure WEST

This section presents the following topics:

- Enabling VPN on WEST
- Configuring an IKE Group on WEST
- Configuring an ESP Group on WEST
- Creating the Connection to EAST

This section presents the following examples:

- Example 14-1 Enabling IPsec VPN on WEST
- Example 14-2 Configuring an IKE group on WEST
- Example 14-3 Configuring an ESP group on Vyatta router WEST
- Example 14-4 Creating a site-to-site connection from WEST to EAST

Enabling VPN on WEST

In this section, you enable IPsec VPN on the interfaces that will be used in VPN connections. The VPN tunnel in the example configuration extends from eth1 on WEST through the wide area network to eth0 on EAST. This means that eth1 on WEST must have VPN enabled. The other interfaces on WEST need not.

To create an IPsec connection with another VPN gateway, you must specify the local IP address to be used as the source IP in packets sent to the destination gateway. This IP address:

- Must be one that is defined on a local Ethernet interface, and
- The interface must have IPsec VPN enabled on it

Example 14-1 enables IPsec VPN on eth1 on WEST. To do this, perform the following steps on WEST in configuration mode:

Example 14-1 Enabling IPsec VPN on WEST

Step	Command
Enable VPN on eth1 on WEST.	`vyatta@WEST# set vpn ipsec ipsec-interfaces interface eth1` `[edit]`

View IPsec interface configuration. Don't commit yet.	`vyatta@WEST# show vpn ipsec ipsec-interfaces`
	`>     interface eth1`
	`[edit]`

Configuring an IKE Group on WEST

The IKE group allows you to pre-define a set of one or more proposals to be used in IKE Phase 1 negotiation, after which the ISAKMP security association (SA) can be set up. For each proposal in the group, the following information is defined:

- The cipher to be used to encrypt packets during IKE Phase 1
- The hash function to be used to authenticate packets during IKE Phase 1

The IKE group also has a configured lifetime, which is the duration of the ISAKMP SA. When the lifetime of the ISAKMP SA expires, a new Phase 1 negotiation takes place, and new encryption, hash, and keying information is established in a new pair of ISAKMP SAs.

The lifetime is an attribute of the IKE group as a whole. If the IKE group contains multiple proposals, the lifetime applies regardless of which proposal in the group is accepted.

Example 14-2 creates IKE group IKE-1W on WEST. This IKE group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses AES-128 as the encryption cipher and SHA-1 as the hash algorithm

The lifetime of a proposal from this IKE group is set to 3600 seconds.

To create this IKE group, perform the following steps on WEST in configuration mode:

Example 14-2 Configuring an IKE group on WEST

Step	Command
Create the configuration node for proposal 1 of IKE group IKE-1W.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of IKE group IKE-1W.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W proposal 2 encryption aes128` `[edit]`
Set the hash algorithm for proposal 2.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W proposal 2 hash sha1` `[edit]`
Set the lifetime for the whole IKE group.	`vyatta@WEST# set vpn ipsec ike-group IKE-1W lifetime 3600` `[edit]`

Example 14-2 Configuring an IKE group on WEST

View the configuration for the IKE group. Don't commit yet.

```
vyatta@WEST# show -all vpn ipsec ike-group IKE-1W
>     proposal 1 {
>        encryption: "aes256"
>        hash: "sha1"
>     }
>     proposal 2 {
>        encryption: "aes128"
>        hash: "sha1"
>     }
>     lifetime: 3600

[edit]
```

Configuring an ESP Group on WEST

Encapsulated Security Payload (ESP) is an authentication protocol that provides authentication for IP packets, and it also encrypts them.

The ESP protocol negotiates a unique number for the session connection, called the Security Parameter Index (SPI). It also starts a numbering sequence for the packets and negotiates the hashing algorithm that will be used to authenticate packets.

The Vyatta router allows you to pre-define multiple ESP configurations. Each one is known as an "ESP group." ESP group includes the Phase 2 proposals, which contain the parameters needed to negotiate an IPsec security association:

- The cipher to be used to encrypt user data across the IPsec tunnel
- The hashing function to be used to authenticate packets in the IPsec tunnel
- The lifetime of the IPsec security association

Example 14-3 creates ESP group ESP-1W on Vyatta router WEST. This ESP group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses Triple-DES as the encryption cipher and MD5 as the hash algorithm

The lifetime of a proposal from this ESP group is set to 1800 seconds.

To create this ESP group, perform the following steps on WEST in configuration mode:

Example 14-3 Configuring an ESP group on Vyatta router WEST

Step	Command
Create the configuration node for proposal 1 of ESP group ESP-1W.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of ESP group ESP-1W.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W proposal 2 encryption 3des` `[edit]`
Set the hash algorithm for proposal 2.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W proposal 2 hash md5` `[edit]`
Set the lifetime for the whole ESP group.	`vyatta@WEST# set vpn ipsec esp-group ESP-1W lifetime 1800` `[edit]`

View the configuration for the ESP group. Don't commit yet.	`vyatta@WEST# show -all vpn ipsec esp-group ESP-1W`
	`>  proposal 1 {`
	`>  encryption: "aes256"`
	`>  hash: "sha1"`
	`>  }`
	`>  proposal 2 {`
	`>  encryption: "3des"`
	`>  hash: "md5"`
	`>  }`
	`>  lifetime: 1800`
	
	`[edit]`

Creating the Connection to EAST

In defining a site-to-site connection, you specify IPsec policy information (most of which is pre-configured as an IKE and ESP group) and the routing information for the two endpoints of the IPsec tunnel.

The local endpoint is the Vyatta router. The remote endpoint is the peer VPN gateway—this can be another Vyatta router, or it can be another IPsec-compliant router, an IPsec-capable firewall, or a VPN concentrator. For each end of the tunnel, you define the IP address and subnet mask of the local and remote subnets or hosts.

In all, you must specify:

- The IP address of the remote peer.
- The authentication mode that the peers will use to authenticate one another. Currently, the Vyatta router supports peer authentication by pre-shared secret (pre-shared key, or PSK), so you must also supply the string that will be used to generate the hashed key.
- The IKE group to be used in the connection.
- The ESP group to be used in the connection.
- The IP address on this Vyatta router to use for the tunnel. This IP address must be pre-configured on the interface enabled for VPN.
- The communicating subnet or host for each end of the tunnel. You can define multiple tunnels for each VPN peer, and each tunnel can use a different security policy.

When supplying a preshared secret, keep the following in mind:

A pre-shared secret, or pre-shared key (PSK), is a method of authentication. The secret, or key, is a string agreed upon beforehand by both parties as key for authenticating the session. It is used to generate a hash such that each VPN endpoint can authenticate the other.

Note that the pre-shared secret, although an ordinary string, is not a “password.” It is actually used to generate a hashed key to form a “fingerprint” proving the identity of each endpoint. This means that long complex strings are more secure than short strings. Choose complex pre-shared secrets and avoid short ones, which can be more easily compromised by an attack.

The preshared secret is not passed during IKE negotiation. It is configured on both sides, and must match on both sides.

A pre-shared secret is an example of *symmetric cryptography:* the key is the same on both sides. Symmetric encryption algorithms are less computationally intensive than asymmetric algorithms, and are therefore faster. However, in symmetric cryptography, the two communicating parties must exchange keys in advance. Doing this securely can be a problem.

Pre-shared secret and digital signatures are the most common methods of IKE authentication. Pre-shared secret is an easy and effective way to quickly set up authentication with little administrative overhead. However, it has several drawbacks.

- If a pre-shared key is captured and no one is aware of it, the attacker has access to your network as long as that key is in use.
- Pre-shared secrets are manually configured, so they should be regularly changed. However, this task is often falls off the list of busy network administrators. Using pre-shared key values with remote users is equivalent to gicing them a password to your network.

NOTE *You should restrict the use of pre-shared keys to smaller, low-risk environments.*

Example 14-4 defines a site-to-site connection to EAST.

- This connection is configured with a single tunnel:
  - Tunnel 1 communicates between 192.168.40.0/24 on WEST and 192.168.60.0/24 on EAST, using ESP group ESP-1W.
- WEST uses IP address 192.0.2.1 on eth1.
- EAST uses IP address 192.0.2.33 on eth0.
- The IKE group is IKE-1W
- The authentication mode is pre-shared secret. The pre-shared secret is “test_key_1”.

To configure this connection, perform the following steps on Vyatta router WEST in configuration mode:

Example 14-4 Creating a site-to-site connection from WEST to EAST

Step	Command
Create the node for EAST and set the authentication mode.	`vyatta@WEST# set vpn ipsec site-to-site peer 192.0.2.33 authentication mode pre-shared-secret` `[edit]`
Navigate to the node for the peer for easier editing.	`vyatta@WEST# edit vpn ipsec site-to-site peer 192.0.2.33` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the string that will be used to generate encryption keys.	`vyatta@WEST# set authentication pre-shared-secret test_key_1` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the IKE group.	`vyatta@WEST# set ike-group IKE-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Identify the IP address on this Vyatta router to be used for this connection.	`vyatta@WEST# set local-ip 192.0.2.1` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create a tunnel configuration, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 1 local-subnet 192.168.40.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`

Example 14-4 Creating a site-to-site connection from WEST to EAST

Provide the remote subnet for the tunnel.	`vyatta@WEST# set tunnel 1 remote-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for this tunnel.	`vyatta@WEST# set tunnel 1 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Return to the top of the configuration tree.	`vyatta@WEST# top` `[edit]`

Now commit the configuration.	`vyatta@WEST# commit`
	`OK`
	`[edit]`

View the configuration for the site-to-site connection.	`vyatta@WEST# show -all vpn ipsec site-to-site peer 192.0.2.33`
	`authentication`
	`mode: pre-shared-secret`
	`pre-shared-secret: "test_key_1"`
	`}`
	`ike-group: IKE-1W`
	`local-ip: 192.0.2.1`
	`tunnel 1 {`
	`local-subnet: 192.168.40.0/24`
	`remote-subnet: 192.168.60.0/24`
	`esp-group: ESP-1W`
	`}`
	
	`[edit]`

Configure EAST

This section presents the following topics:

- Enabling VPN on EAST
- Configuring an IKE Group on EAST
- Configuring an ESP Group on EAST
- Creating the Connection to WEST

This section presents the following examples:

- Example 14-5 Enabling IPsec VPN on EAST
- Example 14-6 Configuring an IKE group on EAST
- Example 14-7 Configuring an ESP group on EAST
- Example 14-8 Creating a site-to-site connection from EAST to WEST

Enabling VPN on EAST

In this section, you enable IPsec VPN on the interfaces that will be used in VPN connections on Vyatta router EAST. The VPN tunnel in the example configuration extends from eth1 on WEST through the wide area network to eth0 on EAST. This means that eth0 on EAST must have VPN enabled. The other interfaces on EAST need not.

Example 14-5 enables IPsec VPN on eth0 on EAST. To do this, perform the following steps on EAST in configuration mode:

Example 14-5 Enabling IPsec VPN on EAST

Step	Command
Enable VPN on eth0 on EAST.	`vyatta@EAST# set vpn ipsec ipsec-interfaces interface eth0` `[edit]`

View IPsec interface configuration. Don't commit yet.	`vyatta@EAST# show vpn ipsec ipsec-interfaces`
	`>     interface eth0`
	`[edit]`

Configuring an IKE Group on EAST

Example 14-6 creates IKE group IKE-1E on EAST. This IKE group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses AES-128 as the encryption cipher and SHA-1 as the hash algorithm

The lifetime of a proposal from this IKE group is set to 3600.

Note that these parameters correspond to those set in IKE-1W on WEST. You must ensure, in defining proposals, that the encryption ciphers and hash algorithms are such that the two peers will be able to agree on at least one combination.

To create this IKE group, perform the following steps on EAST in configuration mode:

Example 14-6 Configuring an IKE group on EAST

Step	Command
Create the configuration node for proposal 1 of IKE group IKE-1E.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E proposal 1 encryption aes256` `[edit]`

Example 14-6 Configuring an IKE group on EAST

Set the hash algorithm for proposal 1.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of IKE group IKE-1E.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E proposal 2 encryption aes128` `[edit]`
Set the hash algorithm for proposal 2.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E proposal 2 hash sha1` `[edit]`
Set the lifetime for the whole IKE group.	`vyatta@EAST# set vpn ipsec ike-group IKE-1E lifetime 3600` `[edit]`

View the configuration for the IKE group. Don't commit yet.	`vyatta@EAST# show -all vpn ipsec ike-group IKE-1E`
	`>  proposal 1 {`
	`>  encryption: "aes256"`
	`>  hash: "sha1"`
	`>  }`
	`>  proposal 2 {`
	`>  encryption: "aes128"`
	`>  hash: "sha1"`
	`>  }`
	`>  lifetime: 3600`
	
	`[edit]`

Configuring an ESP Group on EAST

Example 14-7 creates ESP group ESP-1E on EAST. This ESP group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses Triple-DES as the encryption cipher and MD5 as the hash algorithm

The lifetime of a proposal from this ESP group is set to 1800 seconds.

To create this ESP group, perform the following steps on EAST in configuration mode:

Example 14-7 Configuring an ESP group on EAST

Step	Command
Create the configuration node for proposal 1 of ESP group ESP-1E.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E proposal 1` `[edit]`

Example 14-7 Configuring an ESP group on EAST

Set the encryption cipher for proposal 1.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of ESP group ESP-1E.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E proposal 2 encryption 3des` `[edit]`
Set the hash algorithm for proposal 2.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E proposal 2 hash md5` `[edit]`
Set the lifetime for the whole ESP group.	`vyatta@EAST# set vpn ipsec esp-group ESP-1E lifetime 1800` `[edit]`

View the configuration for the ESP group. Don't commit yet.	`vyatta@EAST# show -all vpn ipsec esp-group ESP-1E`
	`>  proposal 1 {`
	`>  encryption: "aes256"`
	`>  hash: "sha1"`
	`>  }`
	`>  proposal 2 {`
	`>  encryption: "3des"`
	`>  hash: "md5"`
	`>  }`
	`>  lifetime: 1800`
	
	`[edit]`

Creating the Connection to WEST

Example 14-8 defines a site-to-site connection to WEST. In this example:

- This connection is configured with a single tunnel:
  - Tunnel 1 communicates between 192.168.60.0/24 on EAST and 192.168.40.0/24 on WEST, using ESP group ESP-1E.
- EAST uses IP address 192.0.2.33 on eth0.
- WEST uses IP address 192.0.2.1 on eth1.
- The IKE group is IKE-1E.
- The authentication mode is pre-shared secret. The pre-shared secret is "test_key_1".

To configure this connection, perform the following steps on EAST in configuration mode:

Example 14-8 Creating a site-to-site connection from EAST to WEST

Step	Command
Create the node for WEST and set the authentication mode	vyatta@EAST# **set vpn ipsec site-to-site peer 192.0.2.1 authentication mode pre-shared-secret** [edit]
Navigate to the node for the peer for easier editing	vyatta@EAST# **edit vpn ipsec site-to-site peer 192.0.2.1** [edit vpn/ipsec/site-to-site/peer/172.3.3.5]
Provide the string that will be used to generate encryption keys.	vyatta@EAST# **set authentication pre-shared-secret test_key_1** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the IKE group.	vyatta@EAST# **set ike-group IKE-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Identify the IP address on this Vyatta router to be used for this connection.	vyatta@EAST# **set local-ip 192.0.2.33** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Create a tunnel configuration, and provide the local subnet for this tunnel.	vyatta@EAST# **set tunnel 1 local-subnet 192.168.60.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Provide the remote subnet for the tunnel.	vyatta@EAST# **set tunnel 1 remote-subnet 192.168.40.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the ESP group for this tunnel.	vyatta@EAST# **set tunnel 1 esp-group ESP-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Return to the top of the configuration tree.	vyatta@EAST# **top** [edit]
Now commit the configuration.	vyatta@EAST# **commit** OK [edit]

Example 14-8 Creating a site-to-site connection from EAST to WEST

View the configuration for the site-to-site connection.

```
vyatta@EAST# show -all vpn ipsec site-to-site peer
192.0.2.1
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_1"
      }
      ike-group: IKE-1E
      local-ip: 192.0.2.33
      tunnel 1 {
         local-subnet: 192.168.60.0/24
         remote-subnet: 192.168.40.0/24
         esp-group: ESP-1E
      }

[edit]
```

Authenticating with RSA Digital Signatures

This section presents the following topics:

- Generate a Digital Signature on WEST
- Generate a Digital Signature on EAST
- Record EAST's Public Key on WEST
- Modify WEST's Connection to EAST
- Record WEST's Public Key on EAST
- Modify EAST's Connection to WEST

In this set of examples, you modify the VPN connection configured in the previous set of examples between WEST and EAST ("Configuring a Basic Site-to-Site Connection" on page 206). The site-to-site connection created in that set of examples used pre-shared keys for authentication. This set of examples modifies the connection to use RSA digital signatures for authentication.

Generate a Digital Signature on WEST

In this example, you generate WEST's digital signature. This signature will have two parts: a public part (the public key) and a private part (the private key). The public key will be shared with EAST; the private key will remain secret.

To generate an RSA digital signature for router WEST, perform the following steps in operational mode.

Example 14-9 Generating a digital signature on WEST

Step	Command
Generate the key.	`vyatta@WEST> vpn rsa-key generate`
The system warns you that the existing RSA key file will be overwritten. You have the opportunity to exit the key generation process by pressing <Ctrl>+c.	`A local RSA key file already exists and will be overwritten` `<CTRL>C to exit: 8`
The system indicates the location of the file where the key will be written.	`Generating rsa-key to /etc/ipsec.d/rsa-keys/localhost.key`

The system displays the public portion of the generated key.
By default, this key (including the private portion of the key) is stored in /etc/ipsec.d/rsa-keys/localhost.key
	`Your new local RSA key has been generated`
	`The public portion of the key is:`
	
	`0sAQPEOQvukvkv1ofuO8gEKp7IFFZz4lQqMZyVMInoQKUU/T0iKSK/0`
	`NSH9Ldrr8yQUFayzKag6wM7ASXWXKyt0LS1Gn8tJVsjKGaOkFgLREtV`
	`JD3pRzoc7DSUOBViCD6f/TloTkPepRUtW1bmYev2H7tajSO0K0`
	`rqu+7nlocZI0ppMAyF6CS+Wd5W1JBpVGL+EkKfyEl9RagKxRW82XJbg`
	`Y4LG77K2YDN90Wd2GgMY3kf+YJLIzFEt/xRbh2/380FMpdaUYcbY31o`
	`/5PedUutJCK5RMwl+IJGaxrKf1OmCQfzXlkM09ijZx8kzPIlBk`
	`5hulZrbUWjzBJdFcwFAyPM3yCuv3+ndFX00t3ZLfKu+/wX595J`
	
	`vyatta@WEST>`

Generate a Digital Signature on EAST

In this example, you generate EAST's digital signature. This signature will have two parts: a public part (the public key) and a private part (the private key). The public key will be shared with WEST; the private key will remain secret.

To generate an RSA digital signature for router EAST, perform the following steps in operational mode.

Example 14-10 Generating a digital signature on EAST

Step	Command
Generate the key.	`vyatta@EAST> vpn rsa-key generate`

Example 14-10 Generating a digital signature on EAST

The system warns you that the existing RSA key file will be overwritten. You have the opportunity to exit the key generation process by pressing <Ctrl>+c.	`A local RSA key file already exists and will be overwritten` `<CTRL>C to exit: 5`
The system indicates the location of the file where the key will be written.	`Generating rsa-key to /etc/ipsec.d/rsa-keys/localhost.key`

The system displays the public portion of the generated key.
By default, this key (including the private portion of the key) is stored in /etc/ipsec.d/rsa-keys/localhost.key
	`Your new local RSA key has been generated`
	`The public portion of the key is:`
	
	`0sAQOVBIJL+rIkpTuwh8FPeceAF0bhgLr++W51bOAIjFbRDbR8gX3Vl`
	`z6wiUbMgGwQxWlYQiqsCeacicsfZx/amlEn9PkSE4e7tqK/JQo40L5C`
	`7gcNM24mup1d+0WmN3zLb9Qhmq5q3pNJxEwnVbPPQeIdZMJxnb1+lA8`
	`DPC3SIxJM/3at1/KrwqCAhX3QNFY/zNmOtFogELCeyl4+d54wQljA+3`
	`dwFAQ4bboJ7YIDs+rqORxWd3l3I7IajT/pLrwr5eZ8OA9NtAedbMiCw`
	`xyuyUbznxXZ8Z/MAi3xjL1pjYyWjNNiOij82QJfMOrjoXVCfcPn96ZN`
	`+Jqk+KknoVeNDwzpoahFOseJREeXzkw3/lkMN9N1`
	
	`vyatta@EAST>`

Record EAST's Public Key on WEST

In this example, you record the public key you have obtained from EAST. The key is then saved under a name that you can refer to in site-to-site configuration.

A digital signature can be typed in manually, but digital signatures are lengthy and difficult to type. It is generally easier to copy the digital signature into the clipboard of your system and then paste it into the configuration. You do this in a number of ways; for example:

- Receive the public key from the operator of the VPN peer in an e-mail—perhaps an e-mail protected by a PGP signature. Copy the key text into your clipboard.
- From an X.509 certificate, provided by a Certificate Agency.
- Connect to the VPN peer directly through a Telnet or SSH control session. View view the public portion of the key using a **show** command, select the text, and copy the key text into your clipboard.

Example 14-11 pastes EAST's public key into RSA configuration. The name "EAST-key" is used as the identifier of the key.

Before you begin, copy EAST's public key into your clipboard.

If you are in operational mode on WEST, enter configuration mode now and perform the following steps:

Example 14-11 Record EAST's public key on WEST

Step	Command
Specify a name for EAST's public key and paste EAST's public key into the configuration.	`vyatta@WEST# set vpn rsa-keys rsa-key-name EAST-key rsa-key 0sAQOVBIJL+rIkpTuwh8FPeceAF0bhgLr++W51bOAIjFb RDbR8gX3Vlz6wiUbMgGwQxWlYQiqsCeacicsfZx/amlEn9PkSE4e7tq K/JQo40L5C7gcNM24mup1d+0WmN3zLb9Qhmq5q3pNJxEwnVbPPQeIdZ MJxnb1+lA8DPC3SIxJM/3at1/KrwqCAhX3QNFY/zNmOtFogELCeyl4+ d54wQljA+3dwFAQ4bboJ7YIDs+rqORxWd3l3I7IajT/pLrwr5eZ8OA9 NtAedbMiCwxyuyUbznxXZ8Z/MAi3xjL1pjYyWjNNiOij82QJfMOrjoX VCfcPn96ZN+Jqk+KknoVeNDwzpoahFOseJREeXzkw3/lkMN9N1 [edit]`
Commit the configuration.	`vyatta@WEST# commit OK [edit]`
View the configuration for RSA keys. Since you have not changed the configuration for the local host's key, it does not display.	`vyatta@WEST# show vpn rsa-keys` `rsa-key-name EAST-key {` ` rsa-key: "0sAQOVBIJL+rIkpTuwh8FPeceAF0bhgLr++ W51bOAIjFbRDbR8gX3Vlz6wiUbMgGwQxWlYQiqsCeacicsfZx/amlEn 9PkSE4e7tqK/JQo40L5C7gcNM24mup1d+0WmN3zLb9Qhmq5q3pNJxEw nVbPPQeIdZMJxnb1+lA8DPC3SIxJM/3at1/KrwqCAhX3QNFY/zNmOtF ogELCeyl4+d54wQljA+3dwFAQ4bboJ7YIDs+rqORxWd3l3I7IajT/pL rwr5eZ8OA9NtAedbMiCwxyuyUbznxXZ8Z/MAi3xjL1pjYyWjNNiOij8 2QJfMOrjoXVCfcPn96ZN+Jqk+KknoVeNDwzpoahFOseJREeXzkw3/lk MN9N1"` ` }`  `[edit]` `vyatta@WEST#`

Modify WEST's Connection to EAST

Example 14-12 modifies the connection from WEST to EAST to use RSA digital signatures for authentication. In this example:

- The authentication mode is changed from pre-shared secret to RSA digital signatures.
- EAST's public key is specified as the remote key, under the identifier configured in the previous step (see "Record EAST's Public Key on WEST" on page 221).

To modify the site-to-site connection to use RSA configuration, perform the following steps:

Example 14-12 Configure WEST for RSA authentication

Step	Command
Change the authentication mode	vyatta@WEST# **set vpn ipsec site-to-site peer 192.0.2.33 authentication mode rsa** [edit]
Provide the identifier for EAST's digital signature.	vyatta@WEST# **set vpn ipsec site-to-site peer 192.0.2.33 authentication rsa-key-name EAST-key**
Commit the configuration.	vyatta@WEST# **commit** OK [edit]

View the modified configuration for the site-to-site connection.	vyatta@EAST# **show -all vpn ipsec site-to-site peer 192.0.2.33**
	[edit]
	authentication
	mode: "rsa"
	pre-shared-secret: "test_key_1"
	rsa-key-name: "EAST-key"
	}
	ike-group: IKE-1W
	local-ip: 192.0.2.1
	tunnel 1 {
	local-subnet: 192.168.40.0/24
	remote-subnet: 192.168.40.0/24
	esp-group: ESP-1W
	}
	
	[edit]

Record WEST's Public Key on EAST

Example 14-13 pastes WEST's public key into RSA configuration. The name "WEST-key" is used as the identifier of the key.

Before you begin, copy WEST's public key into your clipboard.

If you are in operational mode on EAST, enter configuration mode now and perform the following steps:

Example 14-13 Record WEST's public key on EAST

Step	Command
Specify a name for WEST's public key and paste WEST's public key into the configuration.	`vyatta@EAST# set vpn rsa-keys rsa-key-name WEST-key rsa-key 0sAQPEOQvukvkv1ofuO8gEKp7IFFZz4lQqMZyVMIno QKUU/T0iKSK/0NSH9Ldrr8yQUFayzKag6wM7ASXWXKyt0LS1Gn8tJVs jKGaOkFgLREtVJD3pRzoc7DSUOBViCD6f/TloTkPepRUtW1bmYev2H7 tajSO0K0 rqu+7nlocZI0ppMAyF6CS+Wd5W1JBpVGL+EkKfyEl9RagKxRW82XJbg Y4LG77K2YDN90Wd2GgMY3kf+YJLIzFEt/xRbh2/380FMpdaUYcbY31o /5PedUutJCK5RMwl+IJGaxrKf1OmCQfzXlkM09ijZx8kzPIlBk 5hulZrbUWjzBJdFcwFAyPM3yCuv3+ndFX00t3ZLfKu+/wX595J [edit]`
Commit the configuration.	`vyatta@EAST# commit OK [edit]`
View the configuration for RSA keys. Since you have not changed the configuration for the local host's key, it does not display.	`vyatta@EAST# show vvpn rsa-keys` `rsa-key-name WEST-key {` ` rsa-key: "0sAQPEOQvukvkv1ofuO8gEKp7IFFZz4lQqMZy VMInoQKUU/T0iKSK/0NSH9Ldrr8yQUFayzKag6wM7ASXWXKyt0LS1Gn 8tJVsjKGaOkFgLREtVJD3pRzoc7DSUOBViCD6f/TloTkPepRUtW1bmY ev2H7tajSO0K0 rqu+7nlocZI0ppMAyF6CS+Wd5W1JBpVGL+EkKfyEl9RagKxRW82XJbg Y4LG77K2YDN90Wd2GgMY3kf+YJLIzFEt/xRbh2/380FMpdaUYcbY31o /5PedUutJCK5RMwl+IJGaxrKf1OmCQfzXlkM09ijZx8kzPIlBk 5hulZrbUWjzBJdFcwFAyPM3yCuv3+ndFX00t3ZLfKu+/wX595J"` ` }`  `[edit]` `vyatta@EAST#`

Modify EAST's Connection to WEST

Example 14-12 modifies the connection from EAST to WEST to use RSA digital signatures for authentication.

In this example:

- The authentication mode is changed from pre-shared secret to RSA digital signatures.
- WEST's public key is specified as the remote key, under the identifier configured in the previous step (see "Record WEST's Public Key on EAST" on page 224).

To modify the site-to-site connection to use RSA configuration, perform the following steps:

Example 14-14 Configure EAST for RSA authentication

Step	Command
Change the authentication mode	`vyatta@EAST# set vpn ipsec site-to-site peer 192.0.2.1 authentication mode rsa` `[edit]`
Provide the identifier for WEST's digital signature.	`vyatta@EAST# set vpn ipsec site-to-site peer 192.0.2.1 authentication rsa-key-name WEST-key`

Commit the configuration.	`vyatta@EAST# commit`
	`OK`
	`[edit]`

View the modified configuration for the site-to-site connection.	`vyatta@EAST# show -all vpn ipsec site-to-site peer 192.0.2.1`
	`[edit]`
	`authentication`
	`mode: "rsa"`
	`pre-shared-secret: "test_key_1"`
	`rsa-key: "WEST-key"`
	`}`
	`ike-group: IKE-1E`
	`local-ip: 192.0.2.33`
	`tunnel 1 {`
	`local-subnet: 192.168.60.0/24`
	`remote-subnet: 192.168.40.0/24`
	`esp-group: ESP-1E`
	`}`
	
	`[edit]`

Defining a Connection with NAT

This section presents the following topics:

- Configure WEST
- Configure EAST

Native IPsec packets are encapsulated using Encapsulated Security Payload (ESP). In these packets, the IP addresses are embedded within the encapsulated packet. This causes problems when IPsec packets must traverse a NAT gateway.

When performing Network Address Translation (NAT), the NAT gateway substitutes its own source IP address (and sometimes a port number), for the original source IP and port on outgoing packets. The NAT device listens for a reply, and when a response packet is received, the NAT device reverses the translation so that the incoming packet can arrive at the correct destination. This allows IP addresses within a private network to be “hidden” from external networks.

NAT does not work well with IPsec, because the IP addresses are embedded within the payload of the encapsulated packet. For a number of reasons, this means that the IPsec peer cannot be located behind the NAT device.

The IPsec NAT Traversal protocol (NAT-T, RFCs 3947 and 3948) allows each IPsec packet to be re-encapsulated within a UDP packet, which can be handled correctly by the NAT device. NAT-T runs on top of IPsec. To support NAT-T, the firewall must be set to allow all of the following:

- IKE through UDP port 500
- IPsec NAT-T through UDP port 4500
- ESP

Some gateway devices pre-allow all of these in a feature called “IPsec Passthrough.” However, IPsec Passthrough is incompatible with NAT traversal. IPsec Passthrough devices recognize the IPsec-in-UDP packets and incorrectly attempt passthrough-type operations on the packets. This corrupts the packets in such a way that NAT-T no longer works.

NOTE *If you enable NAT traversal support, make sure you DISABLE IPsec Passthrough on the NAT device.*

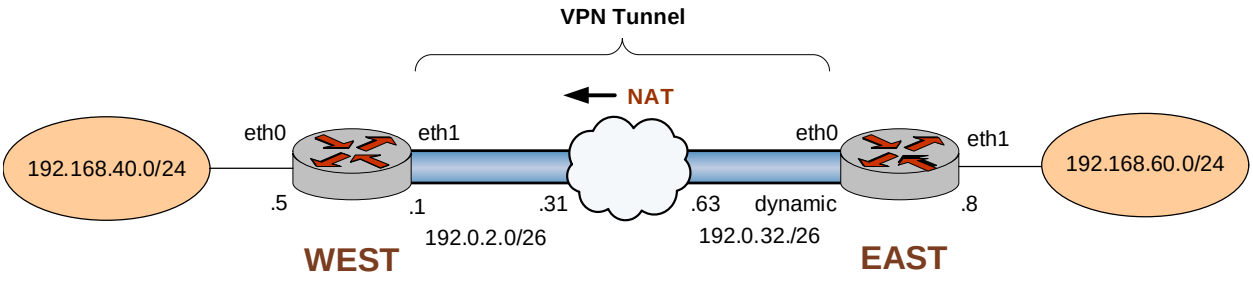
This section presents a sample configuration for a NATted connection between WEST and EAST. In this example:

- EAST resides behind a NAT device, and therefore has a dynamic IP address from WEST’s point of view.
- WEST retains its fixed IP address.

This configuration is similar to something you might see for an IPsec endpoint that is behind a DSL connection, where the DSL peer's public IP address is dynamic and the DSL peer is performing NAT.

When you have finished, these routers will be configured as shown in Figure 14-2.

Figure 14-2 IPsec VPN connection with dynamic IP address and NAT



Before you begin:

- This example assumes that you have already configured a basic site-to-site connection using a preshared key between WEST and EAST, as explained in the section "Configuring a Basic Site-to-Site Connection," which begins on page 157. Only the relevant changes to that configuration are presented here.

Configure WEST

To allow for EAST's dynamic IP address, WEST must create a new site-to-site connection to a peer that has a dynamic IP address.

Example 14-15 defines a new site-to-site connection to EAST.

- The important change is the IP address of the peer. This is set to 0.0.0.0 to represent "any" IP address.
- All other information is set to be the same as the connection created for the basic site-to-site tunnel.

To configure this connection, perform the following steps on WEST in configuration mode:

Example 14-15 Creating a site-to-site connection to a peer with a dynamic IP address

Step	Command
Create the node for EAST, setting the IP address to "any", and set the authentication mode.	`vyatta@WEST# set vpn ipsec site-to-site peer 0.0.0.0 authentication authentication mode pre-shared-secret [edit]`

Example 14-15 Creating a site-to-site connection to a peer with a dynamic IP address

Navigate to the node for the peer for easier editing.	vyatta@WEST# **edit vpn ipsec site-to-site peer 0.0.0.0** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Provide the string that will be used to generate encryption keys.	vyatta@WEST# **set authentication pre-shared-secret test_key_1** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Specify the IKE group.	vyatta@WEST# **set ike-group IKE-1W** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Identify the IP address on this Vyatta router to be used for this connection.	vyatta@WEST# **set local-ip 192.0.2.1** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Create a tunnel configuration, and provide the local subnet for this tunnel.	vyatta@WEST# **set tunnel 1 local-subnet 192.168.40.0/24** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Provide the remote subnet for the tunnel.	vyatta@WEST# **set tunnel 1 remote-subnet 192.168.60.0/24** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Specify the ESP group for this tunnel.	vyatta@WEST# **set tunnel 1 esp-group ESP-1W** [edit vpn/ipsec/site-to-site/peer/0.0.0.0]
Return to the top of the configuration tree.	vyatta@WEST# **top** [edit]
Commit the configuration.	vyatta@WEST# **commit** OK [edit]

View the configuration for the site-to-site connection.	vyatta@WEST# **exit**
	[edit]
	vyatta@WEST# **show -all vpn ipsec site-to-site peer 0.0.0.0**
	authentication
	mode: pre-shared-secret
	pre-shared-secret: "test_key_1"
	}
	ike-group: IKE-1W
	local-ip: 192.0.2.1
	tunnel 1 {
	local-subnet: 192.168.40.0/24
	remote-subnet: 192.168.60.0/24
	esp-group: ESP-1W
	}
	
	[edit]

Configure EAST

The connection from EAST to WEST does not have to be changed in any way from that configured in the section “Configuring a Basic Site-to-Site Connection” on page 206.

- The NAT device keeps track of EAST’s fixed IP and correctly routes incoming packets to EAST, making any necessary changes to outgoing packets
- WEST retains its fixed IP, so no modification is required to the remote peer IP address.

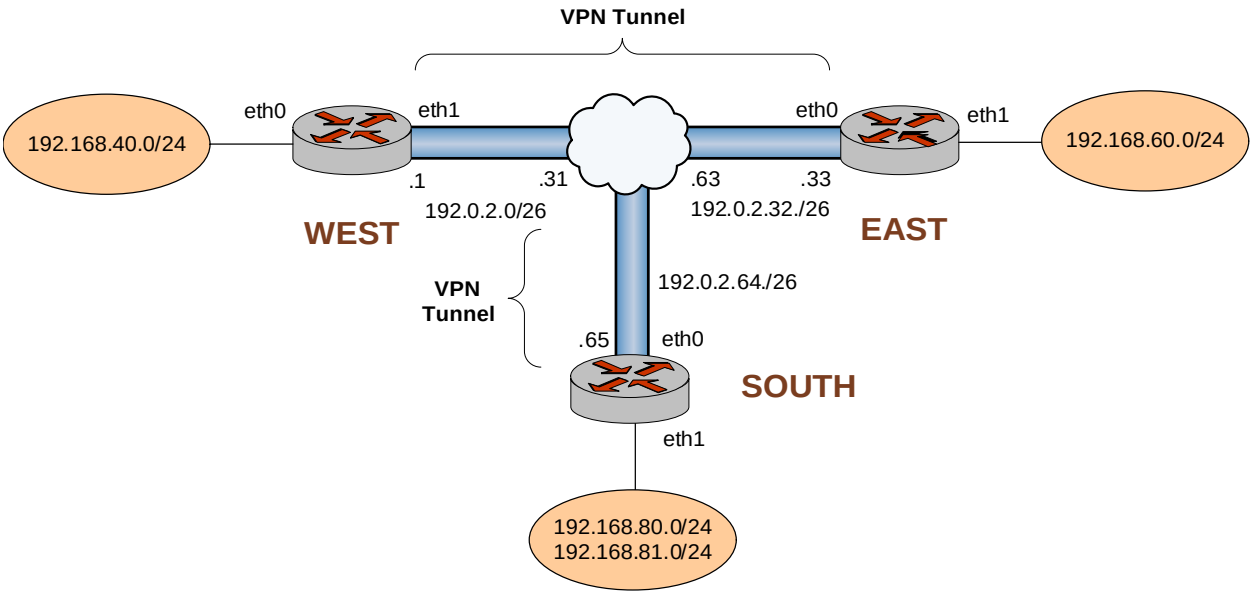
Configuring IPsec Tunnels between Three Gateways

This section presents the following topics:

- Configure WEST
- Configure EAST
- Configure SOUTH

This section presents a sample configuration for multiple site-to-site tunnels between three gateways: WEST, EAST, and SOUTH. When you have finished, these peers will be configured as shown in Figure 14-2.

Figure 14-3 Multiple site-to-site tunnels between three gateways



Configure WEST

This section presents the following topics:

- Configuring the Second ESP Group on WEST
- Adding Tunnels to the Connection to EAST
- Creating the Connection to SOUTH

This example assumes that WEST has already been configured for a basic connection to EAST, as described in “Configuring a Basic Site-to-Site Connection” on page 157. The additional configuration for WEST for this scenario consists of the following:

- An additional ESP group
- Three new tunnel configurations for the site-to-site connection to EAST
- A new site-to-site connection to SOUTH

This section presents the following examples:

- Example 14-16 Configuring a second ESP group on WEST
- Example 14-17 Adding tunnels to the connection to EAST
- Example 14-18 Creating a site-to-site connection from WEST to SOUTH

Configuring the Second ESP Group on WEST

Example 14-16 creates a second ESP group ESP-2W on WEST. This ESP group contains just one proposal:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- The lifetime of a proposal from this ESP group is set to 600 seconds.

To create this ESP group, perform the following steps on WEST in configuration mode:

Example 14-16 Configuring a second ESP group on WEST

Step	Command
Create the configuration node for proposal 1 of ESP group ESP-2W.	`vyatta@WEST# set vpn ipsec esp-group ESP-2W proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@WEST# set vpn ipsec esp-group ESP-2W proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1 of ESP-2W.	`vyatta@WEST# set vpn ipsec esp-group ESP-2W proposal 1 hash sha1` `[edit]`

Example 14-16 Configuring a second ESP group on WEST

Set the lifetime for ESP-2W.	`vyatta@WEST# set vpn ipsec esp-group ESP-2W lifetime 600` `[edit]`

View the configuration for the ESP group. Don't commit yet.	`vyatta@WEST# show -all vpn ipsec esp-group ESP-2W`
	`>     proposal 1 {`
	`>        encryption: "aes256"`
	`>        hash: "sha1"`
	`>     }`
	`>     lifetime: 600`
	
	`[edit]`

Adding Tunnels to the Connection to EAST

Example 14-17 adds three tunnels to the site-to-site connection from WEST to EAST.

- Tunnel 2 communicates between 192.168.40.0/24 on WEST and 192.168.61.0/24 on EAST, and uses ESP group ESP-1W.
- Tunnel 3 communicates between 192.168.41.0/24 on WEST and 192.168.60.0/24 on EAST, and uses ESP group ESP-2W.
- Tunnel 4 communicates between 192.168.41.0/24 on WEST and 192.168.61.0/24 on EAST, and uses ESP group ESP-2W.

To configure this connection, perform the following steps on WEST in configuration mode:

Example 14-17 Adding tunnels to the connection to EAST

Step	Command
Navigate to the configuration node for EAST for easier editing	`vyatta@WEST# edit vpn ipsec site-to-site peer 192.0.2.33` `[edit vpn/ipsec/site-to-site/peer/192.0.2.254]`
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 2 local-subnet 192.168.40.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the remote subnet for tunnel 2.	`vyatta@WEST# set tunnel 2 remote-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 2.	`vyatta@WEST# set tunnel 2 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 3 local-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`

Example 14-17 Adding tunnels to the connection to EAST

Provide the remote subnet for tunnel 3.	`vyatta@WEST# set tunnel 3 remote-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 3.	`vyatta@WEST# set tunnel 3 esp-group ESP-2W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 4 local-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the remote subnet for tunnel 4.	`vyatta@WEST# set tunnel 4 remote-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 4.	`vyatta@WEST# set tunnel 4 esp-group ESP-2W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Return to the top of the configuration tree.	`vyatta@WEST# top` `[edit]`

Commit the configuration.	`vyatta@WEST# commit`
	`OK`
	`[edit]`

Example 14-17 Adding tunnels to the connection to EAST

View the configuration for the site-to-site connection.

```
vyatta@WEST# exit
[edit]
vyatta@WEST> show -all vpn ipsec site-to-site peer
192.0.2.33
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_1"
      }
      ike-group: IKE-1W
      local-ip: 192.2.0.1
      tunnel 1 {
         local-subnet: 192.168.40.0/24
         remote-subnet: 192.168.60.0/24
         esp-group: ESP-1W
      }
      tunnel 2 {
         local-subnet: 192.168.40.0/24
         remote-subnet: 192.168.61.0/24
         esp-group: ESP-1W
      }
      tunnel 3 {
         local-subnet: 192.168.41.0/24
         remote-subnet: 192.168.60.0/24
         esp-group: ESP-2W
      }
      tunnel 4 {
         local-subnet: 192.168.41.0/24
         remote-subnet: 192.168.61.0/24
         esp-group: ESP-2W
      }

[edit]
```

Creating the Connection to SOUTH

Example 14-18 defines a site-to-site connection from WEST to SOUTH.

- The connection has four tunnels:
  - Tunnel 1 communicates between 192.168.40.0/24 on WEST and 192.168.80.0/24 on SOUTH, and uses ESP group ESP-1W.
  - Tunnel 2 communicates between 192.168.40.0/24 on WEST and 192.168.81.0/24 on SOUTH, and uses ESP group ESP-1W.
  - Tunnel 3 communicates between 192.168.41.0/24 on WEST and 192.168.80.0/24 on SOUTH, and uses ESP group ESP-1W.
  - Tunnel 4 communicates between 192.168.41.0/24 on WEST and 192.168.81.0/24 on SOUTH, and uses ESP group ESP-1W.
- WEST uses IP address 192.0.2.1 on eth1.
- SOUTH uses IP address 192.0.2.65 on eth0.
- The IKE group is IKE-1W
- The preshared secret is “test_key_2”.

To configure this connection, perform the following steps on WEST in configuration mode:

Example 14-18 Creating a site-to-site connection from WEST to SOUTH

Step	Command
Create the node for SOUTH and set the authentication mode	`vyatta@WEST# set vpn ipsec site-to-site peer 192.0.2.65 authentication mode pre-shared-secret` `[edit]`
Navigate to the node for the peer for easier editing	`vyatta@WEST# edit vpn ipsec site-to-site peer 192.0.2.65` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Provide the string that will be used to generate encryption keys.	`vyatta@WEST# set authentication pre-shared-secret test_key_2` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the IKE group.	`vyatta@WEST# set ike-group IKE-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Identify the IP address on this Vyatta router to be used for this connection.	`vyatta@WEST# set local-ip 192.0.2.1` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Create the configuration node for tunnel 1, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 1 local-subnet 192.168.40.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`

Example 14-18 Creating a site-to-site connection from WEST to SOUTH

Provide the remote subnet for tunnel 1.	`vyatta@WEST# set tunnel 1 remote-subnet 192.168.80.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the ESP group for tunnel 1.	`vyatta@WEST# set tunnel 1 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 2 local-subnet 192.168.40.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Provide the remote subnet for tunnel 2.	`vyatta@WEST# set tunnel 2 remote-subnet 192.168.81.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the ESP group for tunnel 2.	`vyatta@WEST# set tunnel 2 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 3 local-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Provide the remote subnet for tunnel 3.	`vyatta@WEST# set tunnel 3 remote-subnet 192.168.80.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the ESP group for tunnel 3.	`vyatta@WEST# set tunnel 3 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	`vyatta@WEST# set tunnel 4 local-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Provide the remote subnet for tunnel 4.	`vyatta@WEST# set tunnel 4 remote-subnet 192.168.81.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the ESP group for tunnel 4.	`vyatta@WEST# set tunnel 4 esp-group ESP-1W` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Return to the top of the configuration tree.	`vyatta@WEST# top` `[edit]`

Commit the configuration.	`vyatta@WEST# commit`
	`OK`
	`[edit]`

Example 14-18 Creating a site-to-site connection from WEST to SOUTH

View the configuration for the site-to-site connection.

```
vyatta@WEST# exit
[edit]
vyatta@WEST> show -all vpn ipsec site-to-site peer
192.0.2.65
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_2"
      }
      ike-group: IKE-1W
      local-ip: 192.0.2.1
      tunnel 1 {
         local-subnet: 192.168.40.0/24
         remote-subnet: 192.168.80.0/24
         esp-group: ESP-1W
      }
      tunnel 2 {
         local-subnet: 192.168.40.0/24
         remote-subnet: 192.168.81.0/24
         esp-group: ESP-1W
      }
      tunnel 3 {
         local-subnet: 192.168.41.0/24
         remote-subnet: 192.168.80.0/24
         esp-group: ESP-1W
      }
      tunnel 4 {
         local-subnet: 192.168.41.0/24
         remote-subnet: 192.168.81.0/24
         esp-group: ESP-1W
      }

[edit]
```

Configure EAST

This section presents the following topics:

- Configuring the Second ESP Group on EAST
- Adding Tunnels to the Connection to WEST
- Creating the Connection to SOUTH

This example assumes that EAST has already been configured for a basic connection to WEST, as described in ““Configuring a Basic Site-to-Site Connection” on page 206. The additional configuration for EAST for this scenario consists of the following:

- An additional ESP group
- Three new tunnel configurations for the site-to-site connection to WEST
- A new site-to-site connection to SOUTH

This section presents the following examples:

- Example 14-19 Configuring a second ESP group on EAST
- Example 14-20 Adding tunnels to the connection to WEST
- Example 14-21 Creating a site-to-site connection from EAST to SOUTH

Configuring the Second ESP Group on EAST

Example 14-19 creates a second ESP group ESP-2W on EAST. This ESP group contains just one proposal:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm

The lifetime of a proposal from this ESP group is set to 600 seconds.

To create this ESP group, perform the following steps on EAST in configuration mode:

Example 14-19 Configuring a second ESP group on EAST

Step	Command
Create the configuration node for proposal 1 of ESP group ESP-2E.	`vyatta@EAST# set vpn ipsec esp-group ESP-2E proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@EAST# set vpn ipsec esp-group ESP-2E proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1 of ESP-2E.	`vyatta@EAST# set vpn ipsec esp-group ESP-2E proposal 1 hash sha1` `[edit]`

Example 14-19 Configuring a second ESP group on EAST

Set the lifetime for ESP-2E.	`vyatta@EAST# set vpn ipsec esp-group ESP-2E lifetime 600` `[edit]`

View the configuration for the ESP group. Don't commit yet.	`vyatta@EAST# show -all vpn ipsec esp-group ESP-2E`
	`>     proposal 1 {`
	`>        encryption: "aes256"`
	`>        hash: "sha1"`
	`>     }`
	`>     lifetime: 600`
	
	`[edit]`

Adding Tunnels to the Connection to WEST

Example 14-20 adds three tunnels to the site-to-site connection from EAST to WEST.

- Tunnel 2 communicates between 192.168.60.0/24 on EAST and 192.168.41.0/24 on WEST, and uses ESP group ESP-1E.
- Tunnel 3 communicates between 192.168.61.0/24 on EAST and 192.168.40.0/24 on WEST, and uses ESP group ESP-2E.
- Tunnel 4 communicates between 192.168.61.0/24 on EAST and 192.168.41.0/24 on WEST, and uses ESP group ESP-2E.

To configure this connection, perform the following steps on EAST in configuration mode:

Example 14-20 Adding tunnels to the connection to WEST

Step	Command
Navigate to the configuration node for WEST for easier editing	`vyatta@EAST# edit vpn ipsec site-to-site peer 192.0.2.1` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	`vyatta@EAST# set tunnel 2 local-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Provide the remote subnet for tunnel 2.	`vyatta@EAST# set tunnel 2 remote-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Specify the ESP group for tunnel 2.	`vyatta@EAST# set tunnel 2 esp-group ESP-1E` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	`vyatta@EAST# set tunnel 3 local-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`

Example 14-20 Adding tunnels to the connection to WEST

Provide the remote subnet for tunnel 3.	`vyatta@EAST# set tunnel 3 remote-subnet 192.168.40.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Specify the ESP group for tunnel 3.	`vyatta@EAST# set tunnel 3 esp-group ESP-2E` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	`vyatta@EAST# set tunnel 4 local-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Provide the remote subnet for tunnel 4.	`vyatta@EAST# set tunnel 4 remote-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Specify the ESP group for tunnel 4.	`vyatta@EAST# set tunnel 4 esp-group ESP-2E` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Return to the top of the configuration tree.	`vyatta@EAST# top` `[edit]`

Commit the configuration.	`vyatta@EAST# commit`
	`OK`
	`[edit]`

Example 14-20 Adding tunnels to the connection to WEST

View the configuration for the site-to-site connection.

```
vyatta@EAST# exit
[edit]
vyatta@EAST> show -all vpn ipsec site-to-site peer
192.0.2.1
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_1"
      }
      ike-group: IKE-1E
      local-ip: 192.0.2.33
      tunnel 1 {
         local-subnet: 192.168.60.0/24
         remote-subnet: 192.168.40.0/24
         esp-group: ESP-1E
      }
      tunnel 2 {
         local-subnet: 192.168.60.0/24
         remote-subnet: 192.168.41.0/24
         esp-group: ESP-1E
      }
      tunnel 3 {
         local-subnet: 192.168.61.0/24
         remote-subnet: 192.168.40.0/24
         esp-group: ESP-2E
      }
      tunnel 4 {
         local-subnet: 192.168.61.0/24
         remote-subnet: 192.168.41.0/24
         esp-group: ESP-2E
      }

[edit]
```

Creating the Connection to SOUTH

Example 14-21 defines a site-to-site connection from EAST to SOUTH.

- The connection has four tunnels:
  - Tunnel 1 communicates between 192.168.60.0/24 on EAST and 192.168.80.0/24 on SOUTH, and uses ESP group ESP-1E.
  - Tunnel 2 communicates between 192.168.60.0/24 on EAST and 192.168.81.0/24 on SOUTH, and uses ESP group ESP-1E.
  - Tunnel 3 communicates between 192.168.61.0/24 on EAST and 192.168.80.0/24 on SOUTH, and uses ESP group ESP-1E.
  - Tunnel 4 communicates between 192.168.61.0/24 on EAST and 192.168.81.0/24 on SOUTH, and uses ESP group ESP-1E.
- EAST uses IP address 192.0.2.33 on eth1.
- SOUTH uses IP address 192.0.2.65 on eth0.
- The IKE group is IKE-1E
- The preshared secret is “test_key_2”.

To configure this connection, perform the following steps on EAST in configuration mode:

Example 14-21 Creating a site-to-site connection from EAST to SOUTH

Step	Command
Create the node for SOUTH and set the authentication mode	`vyatta@EAST# set vpn ipsec site-to-site peer 192.0.2.65 authentication mode pre-shared-secret` `[edit]`
Navigate to the node for the peer for easier editing	`vyatta@EAST# edit vpn ipsec site-to-site peer 192.0.2.65` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Provide the string that will be used to generate encryption keys.	`vyatta@EAST# set authentication pre-shared-secret test_key_2` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Specify the IKE group.	`vyatta@EAST# set ike-group IKE-1E` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Identify the IP address on this Vyatta router to be used for this connection.	`vyatta@EAST# set local-ip 192.0.2.33` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`
Create the configuration node for tunnel 1, and provide the local subnet for this tunnel.	`vyatta@EAST# set tunnel 1 local-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.65]`

Example 14-21 Creating a site-to-site connection from EAST to SOUTH

Provide the remote subnet for tunnel 1.	vyatta@EAST# **set tunnel 1 remote-subnet 192.168.80.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Specify the ESP group for tunnel 1.	vyatta@EAST# **set tunnel 1 esp-group ESP-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	vyatta@EAST# **set tunnel 2 local-subnet 192.168.60.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Provide the remote subnet for tunnel 2.	vyatta@EAST# **set tunnel 2 remote-subnet 192.168.81.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Specify the ESP group for tunnel 2.	vyatta@EAST# **set tunnel 2 esp-group ESP-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	vyatta@EAST# **set tunnel 3 local-subnet 192.168.61.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Provide the remote subnet for tunnel 3.	vyatta@EAST# **set tunnel 3 remote-subnet 192.168.80.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Specify the ESP group for tunnel 3.	vyatta@EAST# **set tunnel 3 esp-group ESP-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	vyatta@EAST# **set tunnel 4 local-subnet 192.168.61.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Provide the remote subnet for tunnel 4.	vyatta@EAST# **set tunnel 4 remote-subnet 192.168.81.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Specify the ESP group for tunnel 4.	vyatta@EAST# **set tunnel 4 esp-group ESP-1E** [edit vpn/ipsec/site-to-site/peer/192.0.2.65]
Return to the top of the configuration tree.	vyatta@EAST# **top** [edit]
Commit the configuration.	vyatta@EAST# **commit** OK [edit]

Example 14-21 Creating a site-to-site connection from EAST to SOUTH

View the configuration for the site-to-site connection.

```
vyatta@EAST# exit
[edit]
vyatta@EAST> show -all vpn ipsec site-to-site peer
192.0.2.65
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_2"
      }
      ike-group: IKE-1E
      local-ip: 192.0.2.33
      tunnel 1 {
         local-subnet: 192.168.60.0/24
         remote-subnet: 192.168.80.0/24
         esp-group: ESP-1E
      }
      tunnel 2 {
         local-subnet: 192.168.60.0/24
         remote-subnet: 192.168.81.0/24
         esp-group: ESP-1E
      }
      tunnel 3 {
         local-subnet: 192.168.61.0/24
         remote-subnet: 192.168.80.0/24
         esp-group: ESP-1E
      }
      tunnel 4 {
         local-subnet: 192.168.61.0/24
         remote-subnet: 192.168.81.0/24
         esp-group: ESP-1E
      }

[edit]
```

Configure SOUTH

This section presents the following topics:

- Enabling VPN on SOUTH
- Configuring an IKE Group on SOUTH
- Configuring an ESP Group on SOUTH
- Creating the Connection to WEST
- Creating the Connection to EAST
-

This section presents the following examples:

- Example 14-22 Enabling IPsec VPN on SOUTH
- Example 14-23 Configuring an IKE group on SOUTH
- Example 14-24 Configuring an ESP group on SOUTH
- Example 14-25 Creating a site-to-site connection from SOUTH to WEST
- Example 14-26 Creating a site-to-site connection from SOUTH to EAST

Enabling VPN on SOUTH

In this section, you enable IPsec VPN on the interfaces that will be used in VPN connections on SOUTH. The VPN tunnels in the example configuration extend through the wide-area network to eth0 on SOUTH. This means that eth0 on SOUTH must have VPN enabled. The other interfaces on SOUTH need not.

Example 14-22 enables IPsec VPN on eth0 on SOUTH. To do this, perform the following steps on SOUTH in configuration mode:

Example 14-22 Enabling IPsec VPN on SOUTH

Step	Command
Enable VPN on eth0 on SOUTH.	`vyatta@SOUTH# set vpn ipsec ipsec-interfaces interface eth0` `[edit]`

View IPsec interface configuration. Don't commit yet.	`vyatta@SOUTH# show vpn ipsec ipsec-interfaces`
	`>     interface eth0`
	`[edit]`

Configuring an IKE Group on SOUTH

Example 14-23 creates IKE group IKE-1S on SOUTH. This IKE group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses AES-128 as the encryption cipher and SHA-1 as the hash algorithm

The lifetime of a proposal from this IKE group is set to 3600.

Note that these parameters correspond to those set in IKE-1W on WEST and IKE-1E on EAST. You must ensure, in defining proposals, that the encryption ciphers and hash algorithms are such that the two peers will be able to agree on a combination.

To create this IKE group, perform the following steps on SOUTH in configuration mode:

Example 14-23 Configuring an IKE group on SOUTH

Step	Command
Creates the configuration node for proposal 1 of IKE group IKE-1S.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of IKE group IKE-1S.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S proposal 2 encryption aes128` `[edit]`
Set the hash algorithm for proposal 2.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S proposal 2 hash sha1` `[edit]`
Set the lifetime for the whole IKE group.	`vyatta@SOUTH# set vpn ipsec ike-group IKE-1S lifetime 3600` `[edit]`

Example 14-23 Configuring an IKE group on SOUTH

View the configuration for the IKE group. Don't commit yet.	`vyatta@SOUTH# show -all vpn ipsec ike-group IKE-1S`
	`>  proposal 1 {`
	`>  encryption: "aes256"`
	`>  hash: "sha1"`
	`>  }`
	`>  proposal 2 {`
	`>  encryption: "aes128"`
	`>  hash: "sha1"`
	`>  }`
	`>  lifetime: 3600`
	
	`[edit]`

Configuring an ESP Group on SOUTH

Example 14-24 creates ESP group ESP-1S on SOUTH. This ESP group contains two proposals:

- Proposal 1 uses AES-256 as the encryption cipher and SHA-1 as the hash algorithm
- Proposal 2 uses Triple-DES as the encryption cipher and MD5 as the hash algorithm

The lifetime of a proposal from this ESP group is set to 1800 seconds.

To create this ESP group, perform the following steps on SOUTH in configuration mode:

Example 14-24 Configuring an ESP group on SOUTH

Step	Command
Create the configuration node for proposal 1 of ESP group ESP-1S.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S proposal 1` `[edit]`
Set the encryption cipher for proposal 1.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S proposal 1 encryption aes256` `[edit]`
Set the hash algorithm for proposal 1.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S proposal 1 hash sha1` `[edit]`
Set the encryption cipher for proposal 2. This also creates the configuration node for proposal 2 of ESP group ESP-1S.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S proposal 2 encryption 3des` `[edit]`

Example 14-24 Configuring an ESP group on SOUTH

Set the hash algorithm for proposal 2.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S proposal 2 hash md5` `[edit]`
Set the lifetime for the whole ESP group.	`vyatta@SOUTH# set vpn ipsec esp-group ESP-1S lifetime 1800` `[edit]`

View the configuration for the ESP group. Don't commit yet.	`vyatta@SOUTH# show -all vpn ipsec esp-group ESP-1S`
	`>  proposal 1 {`
	`>  encryption: "aes256"`
	`>  hash: "sha1"`
	`>  }`
	`>  proposal 2 {`
	`>  encryption: "3des"`
	`>  hash: "md5"`
	`>  }`
	`>  lifetime: 1800`
	
	`[edit]`

Creating the Connection to WEST

Example 14-25 defines a site-to-site connection to WEST.

- This connection is configured with four tunnels:
  - Tunnel 1 communicates between 192.168.80.0/24 on SOUTH and 192.168.40.0/24 on WEST, and uses ESP group ESP-1S.
  - Tunnel 2 communicates between 192.168.80.0/24 on SOUTH and 192.168.41.0/24 on WEST, and uses ESP group ESP-1S.
  - Tunnel 3 communicates between 192.168.81.0/24 on SOUTH and 192.168.40.0/24 on WEST, and uses ESP group ESP-1S.
  - Tunnel 4 communicates between 192.168.81.0/24 on SOUTH and 192.168.41.0/24 on WEST, and uses ESP group ESP-1S.
- SOUTH uses IP address 192.0.2.65 on eth0.
- WEST uses IP address 192.0.2.1 on eth1.
- The IKE group is IKE-1S.
- The preshared secret is “test_key_2”.

To configure this connection, perform the following steps on SOUTH in configuration mode:

Example 14-25 Creating a site-to-site connection from SOUTH to WEST

Step	Command
Create the node for WEST and set the authentication mode	vyatta@SOUTH# **set vpn ipsec site-to-site peer 192.0.2.1 authentication mode pre-shared-secret** [edit]
Navigate to the node for the peer for easier editing	vyatta@SOUTH# **edit vpn ipsec site-to-site peer 192.0.2.1** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Provide the string that will be used to generate encryption keys.	vyatta@SOUTH# **set authentication pre-shared-secret test_key_2** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the IKE group.	vyatta@SOUTH# **set ike-group IKE-1S** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Identify the IP address on this Vyatta router to be used for this connection.	vyatta@SOUTH# **set local-ip 192.0.2.65** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Create the configuration node for tunnel 1, and provide the local subnet for this tunnel.	vyatta@SOUTH# **set tunnel 1 local-subnet 192.168.80.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Provide the remote subnet for tunnel 1.	vyatta@SOUTH# **set tunnel 1 remote-subnet 192.168.40.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the ESP group for tunnel 1.	vyatta@SOUTH# **set tunnel 1 esp-group ESP-1S** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	vyatta@SOUTH# **set tunnel 2 local-subnet 192.168.80.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Provide the remote subnet for tunnel 2.	vyatta@SOUTH# **set tunnel 2 remote-subnet 192.168.41.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the ESP group for tunnel 2.	vyatta@SOUTH# **set tunnel 2 esp-group ESP-1S** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	vyatta@SOUTH# **set tunnel 3 local-subnet 192.168.81.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Provide the remote subnet for tunnel 3.	vyatta@SOUTH# **set tunnel 3 remote-subnet 192.168.40.0/24** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]
Specify the ESP group for tunnel 3.	vyatta@SOUTH# **set tunnel 3 esp-group ESP-1S** [edit vpn/ipsec/site-to-site/peer/192.0.2.1]

Example 14-25 Creating a site-to-site connection from SOUTH to WEST

Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	`vyatta@SOUTH# set tunnel 4 local-subnet 192.168.81.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Provide the remote subnet for tunnel 4.	`vyatta@SOUTH# set tunnel 4 remote-subnet 192.168.41.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Specify the ESP group for tunnel 4.	`vyatta@SOUTH# set tunnel 4 esp-group ESP-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.1]`
Return to the top of the configuration tree.	`vyatta@SOUTH# top` `[edit]`

Now commit the configuration.	`vyatta@SOUTH# commit`
	`OK`
	`[edit]`

Example 14-25 Creating a site-to-site connection from SOUTH to WEST

View the configuration for the site-to-site connection.

```
vyatta@SOUTH# exit
[edit]
vyatta@SOUTH> show -all vpn ipsec site-to-site peer
192.0.2.1
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_2"
      }
      ike-group: IKE-1S
      local-ip: 192.0.2.65
      tunnel 1 {
         local-subnet: 192.168.80.0/24
         remote-subnet: 192.168.40.0/24
         esp-group: ESP-1S
      }
      tunnel 2 {
         local-subnet: 192.168.80.0/24
         remote-subnet: 192.168.41.0/24
         esp-group: ESP-1S
      }
      tunnel 1 {
         local-subnet: 192.168.81.0/24
         remote-subnet: 192.168.40.0/24
         esp-group: ESP-1S
      }
      tunnel 1 {
         local-subnet: 192.168.81.0/24
         remote-subnet: 192.168.41.0/24
         esp-group: ESP-1S
      }

[edit]
```

Creating the Connection to EAST

Example 14-26 defines a site-to-site connection to EAST.

- This connection is configured with four tunnels:
  - Tunnel 1 communicates between 192.168.80.0/24 on SOUTH and 192.168.60.0/24 on EAST, and uses ESP group ESP-1S.
  - Tunnel 2 communicates between 192.168.80.0/24 on SOUTH and 192.168.61.0/24 on EAST, and uses ESP group ESP-1S.
  - Tunnel 3 communicates between 192.168.81.0/24 on SOUTH and 192.168.60.0/24 on EAST, and uses ESP group ESP-1S.
  - Tunnel 4 communicates between 192.168.81.0/24 on SOUTH and 192.168.61.0/24 on EAST, and uses ESP group ESP-1S.
- SOUTH uses IP address 192.0.2.65 on eth0.
- EAST uses IP address 192.0.2.33 on eth1.
- The IKE group is IKE-1S.
- The preshared secret is “test_key_2”.

To configure this connection, perform the following steps on SOUTH in configuration mode:

Example 14-26 Creating a site-to-site connection from SOUTH to EAST

Step	Command
Create the node for EAST and set the authentication mode	`vyatta@SOUTH# set vpn ipsec site-to-site peer 192.0.2.33` `[edit]`
Navigate to the node for the peer for easier editing	`vyatta@SOUTH# edit vpn ipsec site-to-site peer 192.0.2.33` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the string that will be used to generate encryption keys.	`vyatta@SOUTH# set authentication pre-shared-secret test_key_2` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the IKE group.	`vyatta@SOUTH# set ike-group IKE-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Identify the IP address on this Vyatta router to be used for this connection.	`vyatta@SOUTH# set local-ip 172.5.5.8` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 1, and provide the local subnet for this tunnel.	`vyatta@SOUTH# set tunnel 1 local-subnet 192.168.80.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`

Example 14-26 Creating a site-to-site connection from SOUTH to EAST

Provide the remote subnet for tunnel 1.	`vyatta@SOUTH# set tunnel 1 remote-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 1.	`vyatta@SOUTH# set tunnel 1 esp-group ESP-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 2, and provide the local subnet for this tunnel.	`vyatta@SOUTH# set tunnel 2 local-subnet 192.168.80.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the remote subnet for tunnel 2.	`vyatta@SOUTH# set tunnel 2 remote-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 2.	`vyatta@SOUTH# set tunnel 2 esp-group ESP-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 3, and provide the local subnet for this tunnel.	`vyatta@SOUTH# set tunnel 3 local-subnet 192.168.81.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the remote subnet for tunnel 3.	`vyatta@SOUTH# set tunnel 3 remote-subnet 192.168.60.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 3.	`vyatta@SOUTH# set tunnel 3 esp-group ESP-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Create the configuration node for tunnel 4, and provide the local subnet for this tunnel.	`vyatta@SOUTH# set tunnel 4 local-subnet 192.168.81.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Provide the remote subnet for tunnel 4.	`vyatta@SOUTH# set tunnel 4 remote-subnet 192.168.61.0/24` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Specify the ESP group for tunnel 4.	`vyatta@SOUTH# set tunnel 4 esp-group ESP-1S` `[edit vpn/ipsec/site-to-site/peer/192.0.2.33]`
Return to the top of the configuration tree.	`vyatta@SOUTH# top` `[edit]`

Now commit the configuration.	`vyatta@SOUTH# commit`
	`OK`
	`[edit]`

Example 14-26 Creating a site-to-site connection from SOUTH to EAST

View the configuration for the site-to-site connection.

```
vyatta@SOUTH# exit
[edit]
vyatta@SOUTH> show -all vpn ipsec site-to-site peer
192.0.2.33
      authentication
         mode: pre-shared-secret
         pre-shared-secret: "test_key_2"
      }
      ike-group: IKE-1S
      local-ip: 192.0.2.54
      tunnel 1 {
         local-subnet: 192.168.80.0/24
         remote-subnet: 192.168.60.0/24
         esp-group: ESP-1S
      }
      tunnel 2 {
         local-subnet: 192.168.80.0/24
         remote-subnet: 192.168.61.0/24
         esp-group: ESP-1S
      }
      tunnel 1 {
         local-subnet: 192.168.81.0/24
         remote-subnet: 192.168.60.0/24
         esp-group: ESP-1S
      }
      tunnel 1 {
         local-subnet: 192.168.81.0/24
         remote-subnet: 192.168.61.0/24
         esp-group: ESP-1S
      }

[edit]
```

Monitoring IPsec VPN

This section presents the following topics:

- IPsec VPN Operational Commands
- Sending IPSec VPN Messages to Syslog

IPsec VPN Operational Commands

You can use the following operational commands to monitor IPsec VPN. Please see the *Vyatta OFR Command Reference* for details on these commands.

Command	Description
`clear vpn ipsec-process`	Restarts the IPsec process.
`show vpn debug`	Provides trace-level information about IPsec VPN.
`show vpn ike rsa-keys`	Displays all configured and generated RSA digital signatures.
`show vpn ike sa`	Provides information about all currently active IKE (ISAKMP) security associations.
`show vpn ike secrets`	Displays configured pre-shared secrets.
`show vpn ike status`	Displays summary information about the IKE process.
`show vpn ipsec sa`	Provides information about all active IPsec security associations.
`show vpn ipsec sa statistics`	Display information about active tunnels that have an IPsec security association (SA).
`show vpn ipsec status`	Displays information about the status of IPsec processes.

This section includes the following examples:

- Example 14-27 Viewing IKE security associations
- Example 14-28 Viewing IKE status information
- Example 14-29 Viewing IPsec security associations

- Example 14-30 Viewing IPsec statistics
- Example 14-31 Viewing IPsec status information
- Example 14-32 Viewing IPsec VPN debug information

NOTE *The sample output in these examples may show information unrelated to the sample configurations.*

Showing IKE Information

To see IKE security associations, you can use the **show vpn ike sa** command, as shown in Example 14-27.

Example 14-27 Viewing IKE security associations

```
vyatta@WEST> show vpn ike sa
Local IP        Peer IP         State     Encrypt   Hash     Active L-Time NAT-T
--------        -------         -----     -------   ----     ------ ------ -----
10.6.0.55       10.6.0.57       up        aes128    sha1     454    28800  disab

vyatta@WEST>
```

To see the status of the IKE process, you can use the **show vpn ike status** command, as shown in Example 14-28.

Example 14-28 Viewing IKE status information

```
vyatta@west> show vpn ike status
IKE Process Running

PID: 5832

vyatta@west>
```

Showing IPsec Information

To see IPsec security associations, you can use the **show vpn ipsec sa** command, as shown in Example 14-29.

Example 14-29 Viewing IPsec security associations

```
vyatta@WEST> show vpn ipsec sa
```

```
Peer IP        Dir SPI      Encrypt    Hash       Active Lifetime
-------        --- ---      -------    ----       ------ --------
10.6.0.57       in  bf8ea130 aes128     sha1        565    3600
10.6.0.57       out 5818d99e aes128     sha1        565    3600

vyatta@WEST>
```

To see IPsec statistics, you can use the **show vpn ipsec statistics** command, as shown in Example 14-30.

Example 14-30 Viewing IPsec statistics

```
vyatta@WEST> show vpn ipsec sa statistics
Peer IP           Dir   SRC Network          DST Network          Bytes
-------           ---   -----------          -----------          -----
10.6.0.57         in    0.0.0.0/0            10.7.0.48/28         0(bytes)
10.6.0.57         out   10.7.0.48/28         0.0.0.0/0            0(bytes)

vyatta@WEST>
```

To see the status of the IPsec process, you can use the **show vpn ipsec status** command, as shown in Example 14-31.

Example 14-31 Viewing IPsec status information

```
vyatta@WEST> show vpn ipsec status
IPSec Process Running  PID: 5832

4 Active IPsec Tunnels

IPsec Interfaces:
  eth1    (10.6.0.55)


vyatta@WEST>
```

Viewing IPsec VPN Debug Information

To see more detailed information when you are troubleshooting, you can use the **show vpn debug** command, with or without the **detail** option. Example 14-32 shows the command without the **detail** option.

Example 14-32 Viewing IPsec VPN debug information

```
vyatta@WEST> show vpn debug
000 interface lo/lo ::1
000 interface lo/lo 127.0.0.1
000 interface eth0/eth0 10.1.0.55
000 interface eth1/eth1 10.6.0.55
000 %myid = (none)
000 debug none
000
000algorithmESPencrypt:id=2,name=ESP_DES,ivlen=8,keysizemin=64,keysizema
x=64
000algorithmESPencrypt:id=3,name=ESP_3DES,ivlen=8,keysizemin=192,keysize
max=192
000algorithmESPencrypt:id=7,name=ESP_BLOWFISH,ivlen=8,keysizemin=40,keys
izemax=448
000algorithmESPencrypt:id=11,name=ESP_NULL,ivlen=0,keysizemin=0,keysizem
ax=0
000algorithmESPencrypt:id=12,name=ESP_AES,ivlen=8,keysizemin=128,keysize
max=256
000algorithmESPencrypt:id=252,name=ESP_SERPENT,ivlen=8,keysizemin=128,ke
ysizemax=256
000algorithmESPencrypt:id=253,name=ESP_TWOFISH,ivlen=8,keysizemin=128,ke
ysizemax=256
000algorithmESPauthattr:id=1,name=AUTH_ALGORITHM_HMAC_MD5,keysizemin=128,
keysizemax=128
--More--
```

Sending IPSec VPN Messages to Syslog

This section presents the following examples:

- Example 14-33 Setting VPN log mode and logging severity

The IPsec process generates log messages during operation. You can direct the system to send IPsec log messages to syslog. The result will depend on how the system syslog is configured.

Keep in mind that in the current implementation, the main syslog file **/var/log/messages** reports only messages of severity **warning** and above, regardless of the severity level configured. If you want to configure a different level of severity for log messages (for example, if you want to see debug messages during troubleshooting), you must configure syslog to send messages into a different file, which you define within syslog.

Configuring log modes is optional. When a log mode is not configured, IPsec log messages consist mostly of IPsec startup and shutdown messages. The log modes allow you to direct the system to inspect the IPsec packets and report the results.

Note that some log modes (for example, **all** and **control**) generate several log messages per packet. Using any of these options may severely degrade system performance.

For information about configuring syslog, please refer to "Chapter 16: Logging."

VPN IPsec log messages use standard syslog levels of severity. For information on syslog severities, please see the section "Log Severities" on page 268.

The Vyatta router supports the following logging modes for IPsec VPN.

Table 14-4 IPsec VPN logging modes

Severity	Meaning
raw	Shows the raw bytes of messages.
crypt	Shows the encryption and decryption of messages.
parsing	Shows the structure of input messages.
emitting	Shows the structure of output messages.
control	Shows the decision-making process of the IKE daemon (Pluto).
private	Allows debugging output with private keys.
all	Enables all logging options.

Note that some logging modes (for example, "all") print several messages per packet. Verbose logging modes can cause severe performance degradation.

Example 14-33 configures logging for VPN messages on WEST. In this example:

- Two logging modes are applied:
  - **raw**, which shows the raw bytes of messages
  - **crypt**, which shows the encryption and decryption of messages.
- Severity is set to **warning**, which is more verbose than the default **err**, but might be suitable for some troubleshooting scenarios.
- Facility is set to **daemon**.

To configure logging in this way, perform the following steps on WEST in configuration mode:

Example 14-33 Setting VPN log mode and logging severity

Step	Command
Apply a log mode of raw.	vyatta@WEST# **set vpn ipsec logging log-modes raw** [edit]
Apply a second log mode of crypt.	vyatta@WEST# **set vpn ipsec logging log-modes crypt** [edit]
Set the log severity to warning.	vyatta@WEST# **set vpn ipsec logging level warning** [edit]
Set the log facility to daemon.	vyatta@WEST# **set vpn ipsec logging facility daemon** [edit]
Commit the configuration.	vyatta@WEST# **commit** OK [edit]
View the configuration for logging.	vyatta@WEST# **exit** [edit] vyatta@WEST> **show -all vpn ipsec logging** facility: daemon level: warning log-modes raw log-modes crypt [edit]

Chapter 15: User Authentication

This chapter describes how to set up user accounts and user authentication.

The following topics are covered:

- Authentication Overview
- Creating “Login” User Accounts
- Configuring for a RADIUS Server
- Viewing Authentication Information

Authentication Overview

The Vyatta system supports either of the following options for user account management:

- A local user database (“login” authentication).
- RADIUS authentication server

The system creates two login user accounts by default: user **vyatta** and user **root**. The user account **vyatta** can be deleted, but the user account **root** is protected and cannot be deleted. The default password for each is **vyatta**.

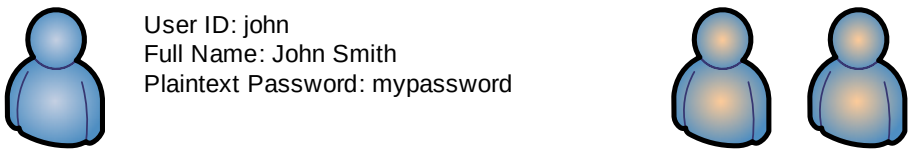
By default, users are authenticated first using the local user database (“login” authentication). If this fails, the system looks for a configured RADIUS server. If found, the router queries the RADIUS server using the supplied RADIUS secret. After the query is validated, the server authenticates the user from information in its database.

You supply login user passwords and RADIUS secrets in plain text. After configuration is committed, the system encrypts them and stores the encrypted version internally. When you display user configuration, only the encrypted version of the password or secret is displayed.

Creating “Login” User Accounts

In this section, a sample configuration is presented for a user account that will be validated using the local user database. The sample configuration used is shown in Figure 15-1.

Figure 15-1 “Login” User Account



This section includes the following example:

- Example 15-1 Creating a “login” user account

Example 15-1 creates a user account for **John Smith**. John has a user ID of **john** and will use a plain text password of **mypassword**. Note that once configuration has been committed, only the encrypted version of the password displays when configuration is shown.

NOTE *User information can be changed through the UNIX shell (providing you have sufficient permissions). However, any changes to Vyatta router user accounts or authentication through the UNIX shell will be overwritten the next time you commit Vyatta router CLI configuration.*

To create a login user account, perform the following steps in configuration mode:

Example 15-1 Creating a "login" user account

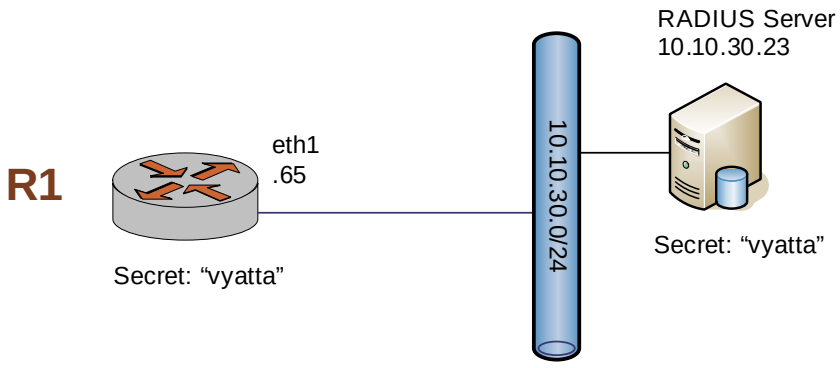
Step	Command
Create the user configuration node, define the user ID, and give the user's full name.	`vyatta@R1# set system login user john full-name "John Smith"` `[edit]`
Specify the user's password in plain text.	`vyatta@R1# set system login user john authentication plaintext-password mypassword` `[edit]`

Commit the change. After a password has been committed, it can be displayed only in encrypted form, as the value of the **encrypted-password** attribute.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Configuring for a RADIUS Server

In this section, a sample configuration is presented for a user account that will be validated using the RADIUS authentication server. The sample configuration used is shown in Figure 15-2.

Figure 15-2 RADIUS User Account



This section includes the following example:

- Example 15-2 Configuring for a RADIUS server

Example 15-2 defines a RADIUS authentication server at 10.10.30.23. The router will access the RADIUS server using a secret of **vyatta**. In this example, the port used for RADIUS traffic is left at the default 1812, which is the well-known port for RADIUS. The timeout after which the next RADIUS server should be queried is left at 2 seconds.

To define a RADIUS server, perform the following steps in configuration mode:

Example 15-2 Configuring for a RADIUS server

Step	Command
Provide the location of the server, and the secret to be used to access it.	`vyatta@R1# set system radius-server 10.10.30.23 secret vyatta` `[edit]`

Commit the change. After the secret has been committed, it can be displayed only in encrypted form.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Viewing Authentication Information

You can always view the information in configuration nodes by using the **show** command in configuration mode. In this case, you can view authentication configuration by using the **show system login** command or the show system radius-server command, as shown in Example 15-3.

To show user authentication configuration, perform the following step in configuration mode:

Example 15-3 Viewing the “system login” and “system radius-server” configuration nodes

Step	Command

Show the contents of the **system login** configuration node.	`vyatta@R1# show system login`
	`user root {`
	`  authentication {`
	`  encrypted-password: “$1$A.EPAdNj$/2bTvc433VZ.VV5YWAbd1"`
	`  }`
	`}`
	`user vyatta {`
	`  authentication {`
	`  encrypted-password: "$1$$ZbzUPUD24iyfRwCKIT16q0"`
	`  }`
	`}`
	`user john {`
	`  full-name: "John Smith"`
	`  authentication {`
	`  encrypted-password: "!!"`
	`  }`
	`  }`

Show the contents of the **system radius-server** configuration node.	`vyatta@R1# show system radius-server`
	`radius-server 10.10.30.23 {`
	`  port: 1812`
	`  secret: “vyatta”`
	`  timeout: 2`
	`}`

Chapter 16: Logging

This chapter explains how the Vyatta system generates and manages system logging messages, and describes how to configure logging.

The following topics are covered:

- Logging Overview
- Enabling and Disabling Logging for Specific Features

Logging Overview

Significant system events are captured in log messages (also called syslog messages), which you can view on the console, save to a file, or forward to an external server such as a syslog server, or direct to the terminal session of one or more specific users.

Depending on the level of message severity you choose to log, system log messages (also called syslog messages) can include notices of ordinary and routine operations, as well as warnings, failure, and error messages.

The Vyatta system's logging function makes use of the UNIX **syslogd** process. Logging configuration performed within the router's CLI is stored in the **/etc/sylogd.conf** file.

By default, local logging is enabled, and sends messages to **/var/log/messages**.

Logging Facilities

The Vyatta system supports standard syslog facilities. These are as follows:

Table 16-1 Syslog facilities

Facility	Description
auth	Authentication and authorization
authpriv	Non-system authorization
cron	Cron daemon
daemon	System daemons
kernel	Kernel
lpr	Line printer spooler
mail	Mail subsystem
mark	Timestamp
news	USENET subsystem
security	Security subsystem
syslog	System logging
user	Application processes
uucp	UUCP subsystem
local0	Local facility 0
local1	Local facility 1

Table 16-1 Syslog facilities

local2	Local facility 2
local3	Local facility 3
local4	Local facility 4
local5	Local facility 5
local6	Local facility 6
local7	Local facility 7
*	All facilities excluding “mark”

In addition, logging can be selectively enabled for some specific routing components. For this information, please see the section ““Enabling and Disabling Logging for Specific Features” on page 269.

Log Destinations

When logging is enabled, system log messages are always written to the **messages** file in the **/var/log** directory of the local file system. In addition, system logs can be sent to the console, to a named file in the local file system, to a server running the **syslogd** utility (that is, a syslog server), or to the terminal session of one or more specific users.

- To direct syslog messages to the console, use the **system syslog console** command.
- To direct syslog messages to a named file in the local file system, use the **system syslog file** command.
- To direct syslog messages to a remote machine running the **syslogd** utility, use the **system syslog host** command.
- To direct syslog messages to the terminal of a specific user, to multiple users, or to all users logged into the routing platform, use the **system syslog user** command.

Log File Locations and Archiving

Messages are written either to the main log file (the default) or to a file that you specify. The main log file is created in the **/var/log** directory, to the **messages** file. User-defined log files are written to the **/var/log/user** directory.

The router uses standard UNIX log rotation to prevent the file system from filling up with log files. When log messages are written to a file, the system will write up to 500 KB of log messages into the file *logfile*, where *logfile* is either the system-defined **messages** file, or a name you have assigned to the file. When *logfile* reaches its maximum size, the system closes it and compresses it into an archive file. The archive file is named *logfile***.0.gz**.

At this point, the logging utility opens a new *logfile* file and begins to write system messages to it. When the new log file is full, the first archive file is renamed *logfile***.1.gz** and the new archive file is named *logfile***.0.gz**.

The system archives log files in this way until a maximum number of log files exists. By default, the maximum number of archived files is 10 (that is, up to *logfile***.9.gz**), where *logfile***.0.gz** always represents the most recent file. After this, the oldest log archive file is deleted as it is overwritten by the next oldest file.

To change the properties of log file archiving, configure the **system syslog archive** node:

- Use the **size** parameter to specify the maximum size of each archived log file.
- Use the **files** parameter to specify the maximum number of archive files to be maintained.

Log Severities

System events generate log messages at different severities, which represent their level of importance for the system.

When you configure severity level for syslog, the system captures log messages at that severity and above. The lower the level of severity specified, the more detail is captured in the logs. For example, if you configure a log severity level of **crit**, the system captures log messages that have severity **crit**, **alert**, and **emerg**.

Currently, log severity is configurable *for user-defined log files only.* The main log file in **/var/log/messages** captures log messages of severity **warning** and above.

Log messages generated by the Vyatta system router will be associated with one of the following levels of severity.

Table 16-2 Syslog message severities

Severity	Meaning
emerg	Emergency. A general system failure or other serious failure has occurred, such that the router is unusable.
alert	Alert. Immediate action is required to prevent the system from becoming unusable—for example, because a network link has failed, or the database has become compromised.
crit	Critical. A critical condition exists, such as resource exhaustion—for example, the system is out of memory, CPU processing thresholds are being exceeded, or a hardware failure has occurred.
err	Error. An error condition has occurred, such as a failed system call. However, the system is still functioning.

Table 16-2 Syslog message severities

warning	Warning. An event has occurred that has the potential to cause an error, such as invalid parameters being passed to a function. This situation should be monitored.
notice	Notice. A normal but significant event has occurred, such as an unexpected event. It is not an error, but could potentially require attention.
info	Informational. Normal events of interest are being reported as they occur.
debug	Debug level. Trace-level information is being provided.

When you specify a severity to report, the system understands that as the *minimum* severity to report, and it reports all log messages of that severity and higher. For example, if you specify a severity of Alert, the system reports all Alert, Critical, and Emergency log messages.

CAUTION *Risk of service degradation. Debug severity is resource-intensive. Setting logging levels to Debug can affect performance.*

Enabling and Disabling Logging for Specific Features

Some features of the Vyatta router—such as BGP, OSPF, and IPsec VPN—produce feature-specific log messages that can be enabled and disabled. When you enable logging for a system feature, the log messages are sent to whatever destinations are configured for syslog.

By default, log messages are sent to the main log file at **/var/log/messages**. You can configure syslog to send log messages to a file you specify in **/var/user**.

- For information about configuring BGP logging, please see “Chapter 10: BGP.”
- For information about configuring OSPF logging, please see “Chapter 9: OSPF.”
- For information about configuring IPsec logging, please see “Chapter 14: IPsec VPN.”

Chapter 17: SNMP

This chapter describes how to configure Simple Network Management Protocol on the Vyatta system.

The following topics are covered:

- SNMP Overview
- Configuring SNMP Information
- Viewing SNMP Information

SNMP Overview

SNMP (Simple Network Management Protocol) is a mechanism for managing network and computer devices.

SNMP uses a manager/agent model for managing the devices. The agent resides in the device, and provides the interface to the physical device being managed. The manager resides on the management system and provides the interface between the user and the SNMP agent. The interface between the SNMP manager and the SNMP agent uses a Management Information Base (MIB) and a small set of commands to exchange information.

MIB Objects

A MIB contains the set of variables/objects that are managed (for example, MTU on a network interface). Those objects are organized in a tree structure where each object is a leaf node. Each object has its unique Object Identifier (OID).

There are two types of objects: *scalar* and *tabular*. A scalar object defines a single object instance. A tabular object defines multiple related object instances that are grouped in MIB tables. For example, the uptime on a device is a scalar object, but the routing table in a router is a tabular object.

Traps

In addition to MIB objects, the SNMP agent on a device can formulate alarms and notifications into SNMP *traps*. The device will asynchronously send the traps to the SNMP managers that are configured as trap destinations or *targets*. This keeps the network manager informed of the status and health of the device.

SNMP Commands

SNMP commands can be used to read or change configuration, or to perform actions on a device, such as resetting it. The set of commands used in SNMP are: **GET**, **GET-NEXT**, **GET-RESPONSE**, **SET**, and **TRAP**.

- **GET** and **GET-NEXT** are used by the manager to request information about an object. These commands are used to view configuration or status, or to poll information such as statistics.
- **SET** is used by the manager to change the value of a specific object. Setting a configuration object changes the device's configuration. Setting an executable object performs an action, such as a file operation or a reset.

- **GET-RESPONSE** is used by the SNMP agent on the device to return the requested information by **GET** or **GET-NEXT**, or the status of the **SET** operation.
- The **TRAP** command is used by the agent to asynchronously inform the manager about events important to the manager.

SNMP Versions

Currently there are three versions of SNMP:

- SNMP v1. This is the first version of the protocol. It is described in RFC 1157.
- SNMP v2. This is an evolution of the first version, and it adds a number of improvements to SNMPv1.
- SNMP v3. This version improves the security model in SNMPv2, and adds support for proxies.

The Vyatta system supports SNMP v2 with community string (SNMP v2c)

SNMP MIB Locations

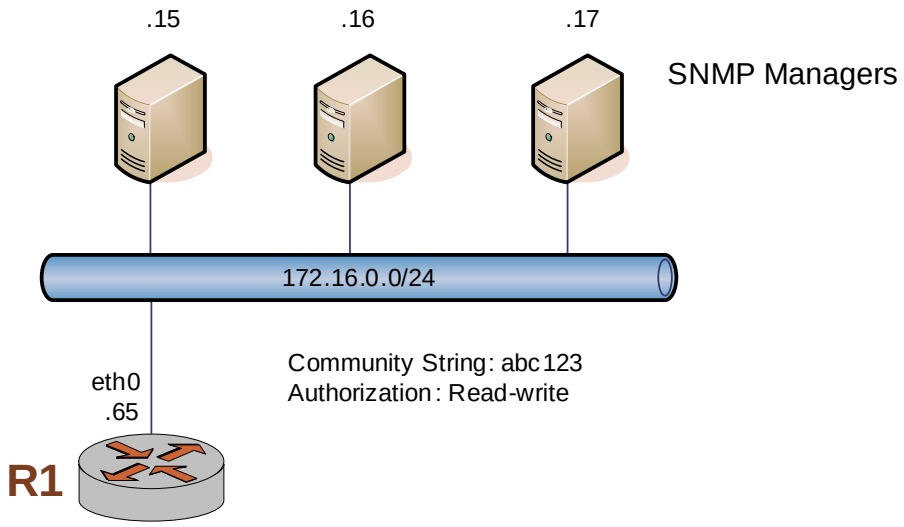
MIBs are typically located in the **/usr/share/snmp/mibs** directory. The Vyatta system does not currently have its own enterprise MIB.

Configuring SNMP Information

To configure SNMP, there must be at least one user created, and the Vyatta MIB model must be loaded.

This sequence sets up an SNMP community that includes three hosts, which will serve as SNMP managers, and configures the router to send traps to all three managers. When you have finished, the router will be configured as shown in Figure 17-1.

Figure 17-1 Configuring SNMP communities and traps



This section includes the following examples:

- Example 17-1 Defining an SNMP community
- Example 17-2 Specifying SNMP trap destinations

Defining the SNMP Community

The SNMP community of a router is the list of SNMP clients authorized to make requests of the router. Authorization for the community is in the form of a community string. The community string acts as a password, providing basic security and protecting the router against spurious SNMP requests.

- If no SNMP clients are explicitly defined, then any client presenting the correct community string is granted read-only access to the router.
- If any client is defined, then only explicitly listed clients are granted access to the router. Those clients will have the access privilege specified by the **authorization** option. (The default is read-only.)

Example 17-1 sets the SNMP community string to abc123 and specifies three clients for the community: 176.16.0.15, 176.16.0.16, and 176.16.0.17. Read-write access is provided for this community.

To define an SNMP community, perform the following steps in configuration mode:

Example 17-1 Defining an SNMP community

Step	Command
Create the **snmp** configuration node and the **community** configuration node. Set the community string. Navigate to the configuration node of the community.	vyatta@R1# **set protocols snmp community abc123** [edit] vyatta@R1# **edit protocols snmp community abc123** [edit protocols snmp community abc123]
List the SNMP clients making up this community.	vyatta@R1# **set client 176.16.0.15** [edit protocols snmp community abc123] vyatta@R1# **set client 176.16.0.16** [edit protocols snmp community abc123] vyatta@R1# **set client 176.16.0.17** [edit protocols snmp community abc123]
Set the privilege level for this community to read-write.	vyatta@R1# **set authorization rw** [edit protocols snmp community abc123]
Commit the change, and return to the top of the configuration tree.	vyatta@R1# **commit** OK [edit protocols snmp community abc123] vyatta@R1# **top** [edit]

Specifying Trap Destinations

Example 17-1 directs the router to send SNMP traps to the configured network managers at 176.16.0.15, 176.16.0.16, and 176.16.0.17.

To specify trap destinations, perform the following steps in configuration mode:

Example 17-2 Specifying SNMP trap destinations

Step	Command

Define the trap destinations, one at a time.	`vyatta@R1# set protocols snmp trap-target 176.16.0.15`
	`[edit]`
	`vyatta@R1# set protocols snmp trap-target 176.16.0.16`
	`[edit]`
	`vyatta@R1# set protocols snmp trap-target 176.16.0.17`
	`[edit]`

Commit the change.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Viewing SNMP Information

You can always view the information in configuration nodes by using the **show** command in configuration mode. In this case, you can view SNMP configuration by using the **show protocols snmp** command, as shown in Example 17-3.

To show the information in the **protocols snmp** configuration node, perform the following step in configuration mode:

Example 17-3 Viewing the “protocols snmp” configuration node

Step	Command

Show the contents of the **protocols snmp** configuration node.	`vyatta@R1# show protocols snmp`
	`community abc123 {`
	`client 176.16.0.15`
	`client 176.16.0.16`
	`client 176.16.0.17`
	`authorization: "rw"`
	`}`

Chapter 18: Software Upgrades

This chapter describes how to use the Vyatta system's package mechanism to update your software.

The following topics are covered:

- Upgrade Overview
- Configuring for Automatic Upgrade
- Working with Packages

Upgrade Overview

Software components updates are stored in *packages*. A package is a precompiled software component that can be installed or upgraded independently of other software components on your machine. Each package contains software and any scripts required to install, configure, or remove it.

To update software packages, you must be running the router shell as user **root**.

Vyatta system packages are stored in the Vyatta software repository.

- The stock repository name for the current release is **vyatta/2.1**.
- The host on which the repository resides is **archive.vyatta.com**.

The Vyatta system keeps a record of which of all available packages have been installed on the router and which have not. The system keeps track of differences between packages, including differences between versions of packages, as necessary. The system also keeps track of dependencies between packages. When you direct the system to install a package, the system will also install any packages on which the specified package depends.

Installation of packages is atomic: all packages, and the packages on which they depend must install successfully, or else the installation will be rolled back, leaving the system as it was before.

Some updates (for example, a kernel upgrade) may require a system reboot, but in general, updates will not require a reboot.

Configuring for Automatic Upgrade

Software update is configured in the **system package** configuration node. To use the automatic upgrade feature, the system must be configured:

- The name of the software repository, and
- The URL of the host on which the repository is to be found.

You can view package configuration by using the **show system package** command in configuration mode, as shown in Example 18-1.

Example 18-1 Viewing the “system package” configuration node

Step	Command

Show the contents of the **system package configuration** node.	`vyatta@R1# show system package`
	`repository 1.1 {`
	`component: "main"`
	`url: "http://archive.vyatta.com"`
	`}`
	
	`[edit]`
	`vyatta@R1#`

If you need to restore package configuration, you can perform the following steps, in configuration mode:

Example 18-2 Configuring for automatic software update

Step	Command
Create the **package** configuration node, specifying the repository and host.	`vyatta@R1# set system package repository 2.1 url http://archive.vyatta.com` `[edit]`
Specify “main” as a component.	`vyatta@R1# set system package repository 2.1 component main` `[edit]`

Commit the configuration.	`vyatta@R1# commit`
	`OK`
	`[edit]`

Working with Packages

To update software, you can use the following workflow:

1. View the List of Available Packages
2. Upgrade Packages

If you want to install a specific package, or remove a package, you can:

3. Install Packages
4. Delete a package

View the List of Available Packages

The router keeps a list of available packages, which it synchronizes with the software repository. However, you can force the system to poll the repository by issuing the **update package-list** command.

We recommend monitoring the list of available packages regularly, so that you can be aware of what updates are available.

Updating the list of packages may take some time to complete, and the system displays a progress indicator. You can cancel the update at any time, by pressing <Ctrl>-c.

To update the list of packages, issue the **update package-list** command in operational mode:

Example 18-3 Updating the package list

```
vyatta@R1> update package-list
```

To view the information in the current package list, issue the **show package info** command in operational mode:

Example 18-4 Viewing package information

```
vyatta@R1> show package info
```

Upgrade Packages

Running this command upgrades software packages on your system, including their dependencies. All packages are upgraded, unless you filter upgrades by specifying package names.

Packages are downloaded from the repository and upgraded in the correct order. Packages are upgraded to the most recent version available in the repository, provided all dependencies can be satisfied. Packages for which dependencies cannot be satisfied, or that have conflicts with installed software, are not “kept back” and not installed.

Before upgrading, you should use the **show package info** command to confirm the complete list of packages that will be upgraded.

To upgrade software packages, issue the **update package** command in operational mode:

Example 18-5 Upgrading software packages

```
vyatta@R1> update package
```

Install Packages

Instead of upgrading your entire system, you can select specific packages to install.If the package has a dependency, the necessary packages will also be installed.

To upgrade software packages, issue the **install package** command in operational mode. Example 18-6 installs any package matching the string **grub-man**.

Example 18-6 Installing specific packages

```
vyatta@R1> install package grub-man
```

Removing a Package

You can remove any unwanted packages from the system. When you remove a package, any packages that depend on it are also removed. You cannot remove a package without removing packages that depend on it. Although the packages are removed, the configuration files (if any) will remain in the system.

To remove a software package, issue the **delete package** command in operational mode. Example 18-7 removes any package matching the string **grub-man**.

Example 18-7 Removing a package

```
vyatta@R1> delete package grub-man
```

Quick Guide to Configuration Statements

Use this section to quickly see the complete syntax of configuration statements.

The Vyatta system supports the following configuration statements:

- firewall
- interfaces
- multicast
- policy
- protocols
- rtrmgr
- service
- system
- vpn

firewall

```
firewall {
   log-martians: [enable|disable]
   send-redirects: [enable|disable]
   receive-redirects: [enable|disable]
   ip-src-route: [enable|disable]
   broadcast-ping: [enable|disable]
   syn-cookies: [enable|disable]
   name: text {
      description: text
      rule: 1–1024 {
         protocol: [all|tcp|udp|icmp|igmp|ipencap|gre|esp|ah|
            ospf|pim|vrrp]
         icmp {
            type: text {
            code: text
         }
         state {
            established: [enable|disable]
            new: [enable|disable]
            related: [enable|disable]
            invalid: [enable|disable]
            }
         action: [accept|drop|reject]
         log: [enable|disable]
         source {
            address: ipv4
            network: ipv4net
            range {
               start: ipv4
               stop: ipv4
            }
            port-number: 1–65535
            port-name: [http|ftp|smtp|telnet|ssh|dns|snmp]
            port-range {
               start: 1–65535
               stop: 1–65535
            }
         }
         destination {
            address: ipv4
            network: ipv4net
            range {
               start: ipv4
               stop: ipv4
            }
```

```
                port-number: 1-65535
                port-name: [http|ftp|smtp|telnet|ssh|dns|snmp]
                port-range {
                   start: 1-65535
                   stop: 1-65535
                }
             }
          }
       }
    }
```

interfaces

```
interfaces {
   restore: [true|false]
   loopback lo {
      description: text
      address [ipv4|ipv6]{
         prefix-length: [0–32|0-128]
         broadcast: ipv4
         multicast-capable: [true|false]
         disable: [true|false]
      }
   }
   bridge br0..br9 {
      description: text
      disable: [true|false]
      aging: 1–4294967296
      stp: [true|false]
      priority: 1–4294967296
      forwarding-delay: 1–4294967296
      hello-time: 1–4294967296
      max-age: 1–4294967296
   }
   ethernet eth0..eth23 {
      disable:[true|false]
      discard:[true|false]
      description:text
      mac: mac-addr
      hw-id: mac-addr
      mtu: 68–65535
      duplex: [full|half|auto]
      speed: [10|100|1000|auto]
      address: [ipv4|ipv6]{
         prefix-length: [0–32|0-128]
         broadcast: ipv4
         multicast-capable: [true|false]
         disable: [true|false]
      }
      bridge-group {
         bridge: br0..br9
         cost: 1–4294967296
         priority: 1–4294967296
      }
      vrrp {
         vrrp-group: 1–255
         virtual-address: ipv4
         authentication:text
```

```
            advertise-interval: 1–255
            preempt:[true|false]
            priority: 1–255
         }
         firewall {
            in {
               name: text
            }
            out {
               name: text
            }
            local {
               name: text
            }
         }
         vif 1–4096 {
            disable:[true|false]
            address: [ipv4|ipv6]{
               prefix-length: [0–32|0-128]
               broadcast: ipv4
               multicast-capable: [true|false]
               disable: [true|false]
            }
            bridge-group {
               bridge: br0..br9
               cost: 1–4294967296
               priority: 1–4294967296
            }
            vrrp {
               vrrp-group: 1–255
               virtual-address: ipv4
               authentication:text
               advertise-interval: 1–255
               preempt:[true|false]
               priority: 1–255
            }
            firewall {
               in {
                  name: text
               }
               out {
                  name: text
               }
               local {
                  name: text
               }
            }
      serial [wan0..wan9] {
```

```
encapsulation: [ppp|cisco-hdlc|frame-relay]
description: text
t1-options {
   lbo: [0-110ft|110-220fr|220-330ft|330-440ft|440-550ft]
   timeslots {
      start: [1–24]
      stop: [1–24]
   }
   mtu: 8–8188
   clock: [internal|external]
}
e1-options {
   framing: [g704|g704-no-crc4|unframed]
   timeslots {
      start: [1–32]
      stop: [1–32]
   }
   mtu: 8–8188
   clock: [internal|external]
}
t3-options {
   framing: [c-bit|m13]
   line-coding: [ami|b8zs]
}
ppp {
   authentication {
      type: [none|chap|pap]
      user-id: text
      password: text
   }
   vif 1 {
      address {
         local-address: ipv4
         prefix-length: 0–32
         remote-address: ipv4
      }
      description: text
      firewall {
         in {
            name: text
         }
         out {
            name: text
         }
         local {
            name: text
         }
```

```
        }
    }
}
cisco-hdlc {
    keepalives {
        require-rx: [enable|disable]
        timer: 10-60000
    }
    vif 1 {
        address {
            local-address: ipv4
            prefix-length: 0–32
            remote-address: ipv4
        }
        description: text
        firewall {
            in {
                name: text
            }
            out {
                name: text
            }
            local {
                name: text
            }
        }
    }
}
frame-relay {
    signaling: [auto|ansi|q933|lmi]
    signaling-options {
        n39ldte: 1–255
        n392dte: 1–100
        n393dte: 1–10
        t391dte: 5–30
    }
    vif [16..991] {
        address {
            local-address: ipv4
            prefix-length: 0–32
            remote-address: ipv4
        }
        description: text
        firewall {
            in {
                name: text
            }
```

```
                out {
                   name: text
                }
                local {
                   name: text
                }
             }
          }
       }
       tunneltun0..tun1024 {
          description: text
          local-ip: ipv4
          remote-ip: ipv4
          outbound-interface: [eth0..eth23]
          encapsulation: [gre|ipip]
          address ipv4 {
             prefix-length: 0-32
          }
          ttl: 0-255
          tos: 0-99
          key: 0-999999
          mtu: 64-8024
          firewall {
             in {
                name: text
             }
             out {
                name: text
             }
             local {
                name: text
             }
          }
       }
    }
```

multicast

```
multicast {
    mfea4 {
        disable:bool
        interface: eth0..eth23
        traceoptions {
            flag {
                all {
                    disable:bool
                }
            }
        }
    }
    mfea6 {
        disable:bool
        interface: eth0..eth23
        traceoptions {
            flag {
                all {
                    disable:bool
                }
            }
        }
    }
}
```

policy

```
policy {
   policy-statement: text {
      term: text {
         from {
            protocol: text
            network4: ipv4net
            network6: ipv6net
            network4-list: text
            network6-list: text
            prefix-length4: 0–32-range
            prefix-length6: 0–128-range
            nexthop4: ipv4-range
            nexthop6: ipv6-range
            as-path: text
            as-path-list: text
            community: text
            community-list: text
            neighbor: ipv4-range
            origin: [0|1|2]
            med: int-range
            localpref: int-range
            metric: 1–65535-range
            external: [type-1|type-2]
            tag: int-range
         }
         to {
            network4: ipv4net
            network6: ipv6net
            network4-list: text
            network6-list: text
            prefix-length4: 0–32-range
            prefix-length6: 0–128-range
            nexthop4: ipv4-range
            nexthop6: ipv6-range
            as-path: text
            as-path-list: text
            community: text
            neighbor: ipv4-range
            origin: int
            med: int-range
            localpref: int-range
            was-aggregated: bool
            metric: 1–65535-range
            external: [type-1|type-2]
            tag: int-range
```

```
            }
            then {
               action: [accept|reject]
               trace: int
               nexthop4: next-hop
               nexthop6: ipv6
               as-path-prepend: int
               as-path-expand: int
               community: text
               community-add: text
               community-del: text
               origin: int
               med: int
               med-remove: [true|false]
               localpref: int
               aggregate-prefix-len: int
               aggregate-brief-mode: int
               metric: 1–65535
               external: [type-1|type-2]
               tag: int
            }
         }
      }
      community-list: text {
         elements: text
      }
      community-list: text {
         elements: text
      }
      network6-list: text {
         elements: text
      }
   }
```

protocols

```
protocols
   bgp {
      bgp-id: ipv4
      local-as: 1–65535
      route-reflector {
          cluster-id: ipv4
          disable: [true|false]
      }
      confederation {
          identifier: 1–4294967296
          disable: [true|false]
      }
      damping {
          half-life: 1–4294967296
          max-suppress: 1–4294967296
          reuse: 1–4294967296
          suppress: 1–4294967296
          disable: [true|false]
      }
      peer: text {
         peer-port: 1–4294967296
         local-port: 1–4294967296
         local-ip: text
         as: 1–65535
         next-hop: ipv4
         holdtime: 0,3–65535
         delay-open-time: 1–4294967296
         client: [true|false]
         confederation-member: [true|false]
         prefix-limit {
            maximum: 1–4294967296
            disable: [true|false]
         disable: [true|false]
         ipv4-unicast: [true|false]
         ipv4-multicast: [true|false]
         }
      traceoptions {
         flag {
            verbose {
               disable: [true|false]
            }
            all {
               disable: [true|false]
            }
            message-in {
```

```
                disable: [true|false]
            }
            message-out {
                disable: [true|false]
            }
            state-change {
                disable: [true|false]
            }
            policy-configuration {
                disable: [true|false]
            }
        }
        import: text
        export: text
    }
}
ospf4 {
    router-id: ipv4
    RFC1538Compatibility: [true|false]
    ip-router-alert: [true|false]
    traceoptions {
    flag {
        all {
            disable:[true|false]
        }
    }
    area: ipv4 {
        area-type:[normal|stub|nssa]
        default-lsa {
            disable:[true|false]
            metric: 1–4294967296
        }
        summaries {
            disable:[true|false]
        }
        area-range: ipv4net {
            advertise:[true|false]
        }
        virtual-link: ipv4 {
            transit-area: ipv4
            hello-interval:1–65535
            router-dead-interval: 1–4294967295
            retransmit-interval: 1–65535
            transit-delay:0–3600
            authentication {
                simple-password:text
                md5: 0–255 {
                    password: text
```

```
                    start-time: YYYY-MM-DD.HH:MM
                    end-time: YYYY-MM-DD.HH:MM
                    max-time-drift: 0–65534,65535
                }
            }
        }
        interface: text {
            link-type:[broadcast|p2p|p2m]
            address: ipv4 {
                priority:0–255
                hello-interval:1–65535
                router-dead-interval: 1–4294967296
                interface-cost:1–65535
                retransmit-interval: 1–65535
                transit-delay:0–3600
                authentication {
                    simple-password:text
                    md5: 0–255 {
                        password: text
                        start-time: YYYY-MM-DD.HH:MM
                        end-time: YYYY-MM-DD.HH:MM
                        max-time-drift: 0–65534,65535
                    }
                }
                passive: [true|false]
                neighbor: ipv4 {
                    router-id: ipv4
                }
                disable: [true|false]
            }
        }
    }
    import: text
    export: text
}
rip {
    interface: text {
        address: ipv4 {
            metric: 1–16
            horizon:
                [none|split-horizon|split-horizon-poison-reverse]
            disable: [true|false]
            passive: [true|false]
            accept-non-rip-requests: [true|false]
            accept-default-route: [true|false]
            advertise-default-route: [true|false]
            route-expiry-secs: 1–4294967296
            route-deletion-secs: 1–4294967296
```

```
                triggered-update-min-secs: 1–4294967296
                triggered-update-max-secs: 1–4294967296
                table-announce-min-secs: 1–4294967296
                table-announce-max-secs: 1–4294967296
                table-request-secs: 1–4294967296
                interpacket-delay-msecs: 1–4294967296
                authentication {
                   simple-password: text
                   md5: 0–255 {
                      password: text
                      start-time: YYYY-MM-DD.HH:MM
                      end-time: YYYY-MM-DD.HH:MM
                   }
                }
             }
          }
          import: text
          export: text
       }
       snmp {
          mib-module: text {
             abs-path:text
             mib-index:int
          }
          community: text {
             authorization: [ro|rw]
             client: ipv4 {}
          }
          contact: text
          description: text
          location: text
          trap-target: ipv4 {}
       }
       static {
          disable: [true|false]
          route: ipv4net {
             next-hop: ipv4
             metric: 1–65535
          }
          interface-route: ipv4net {
             next-hop-interface: text
             next-hop-router: ipv4
             metric: 1–65535
          }
          import: text
       }
    }
```

rtrmgr

```
rtrmgr {
    config-directory: text
}
```

service

```
service {
   dhcp-server {
      name text {
         interface: eth0..eth23
         network-mask: 0–32
         start ipv4 {
            stop: ipv4
         }
         exclude: ipv4 {}
         static-mapping: text {
            ip-address: ipv4
            mac-address: macaddr
         }
         dns-server ipv4 {}
         default-router: ipv4
         wins-server ipv4 {}
         lease: 120–4294967296
         domain-name: text
         authoritative: [enable|disable]
      }
   }
   http {
      port: 1–65534
   }
   ssh {
      port: 1–65534
      protocol-version: [v1|v2|all]
   }
   telnet {
      port: 1–65534
   }
   nat {
      rule: 1–1024 {
         type: [source|destination]
         translation-type: [static|dynamic|masquerade]
         inbound-interface: text
         outbound-interface: text
         protocols: [tcp|udp|icmp|all]
         source {
            address: ipv4
            network: ipv4net
            port-number: 1–4294967296 {}
            port-name: [http|ftp|smtp|telnet|ssh|dns|snmp] {}
            port-range {
               start: 1–4294967296
```

```
                stop: 1–4294967296
            }
        }
        destination {
            address: ipv4
            network: ipv4net
            port-number: 1–4294967296 {}
            port-name: [http|ftp|smtp|telnet|ssh|dns|snmp] {}
            port-range {
                start: 1–4294967296
                stop: 1–4294967296
            }
        inside-address {
            address: ipv4
            network: ipv4net
        }
        outside-address {
            address: ipv4
            network: ipv4net
            range {
                start: ipv4
                stop: ipv4
            }
        }
    }
  }
}
```

system

```
system {
   disable: [true | false]
   host-name: text
   domain-name: text
   domain-search {
      domain: text [text ...]
   }
   name-server: ipv4 {}
   time-zone: text
   ntp-server: [ipv4|text] {}
   static-host-mapping {
      host-name: text {
         inet: ipv4
         alias: text {}
      }
   }
   login {
      user text {
         full-name: text
         authentication {
            plaintext-password: text
            encrypted-password: text
         }
      }
      radius-server ipv4 {
         port: 1–65534
         secret: text
         timeout: 1–4294967296
      }
   }
   syslog {
      console {
         facility: text {
            level: text
         }
      }
      global {
         facility: text {
            level: text
         }
         archive {
            files: 1–4294967296
            size: 1–4294967296
         }
      }
```

```
            file text {
               facility: text {
                  level: text
               }
               archive {
                  files: 1–4294967296
                  size: 1–4294967296
               }
            }
            host text {
               facility: text {
                  level: text
               }
            }
            user text {
               facility: text {
                  level: text
               }
            }
         }
         package {
            repository: text {
               description: text
               url: text
               component: text
            }
         }
      }
```

vpn

```
vpn {
   ipsec {
      ipsec-interfaces {
         interface int-name {}
      }
      nat-traversal: [enable|disable]
      nat-networks {
         allowed-network ipv4net {
            exclude ipv4net {}
         }
      }
      copy-tos: [enable|disable]
      ike-group text{
         proposal: 1-65535 {
            encryption: [aes128|aes256|3des]
            hash: [sha1|md5]
            dh-group: [2|5]
         }
         lifetime: 30-86400
         aggressive-mode: [enable|disable]
         dead-peer-detection {
            interval: 15-86400
            timeout: 30-86400
            action: [hold|clear|restart]
         }
      }
      esp-group text {
         proposal 1-65535 {
            encryption: [aes128|aes256|3des]
            hash: [sha1|md5]
         }
         mode: [tunnel|transport]
         lifetime: 30-86400
         pfs: [enable|disable]
         compression: [enable|disable]
      }
      site-to-site {
         peer ipv4 {
            authentication {
               mode: [pre-shared-secret|rsa]
               pre-shared-secret: text
               rsa-key-name: text
            }
            ike-group: text
            local-ip: ipv4
```

```
            tunnel 1-65535 {
               local-subnet: ipv4net
               remote-subnet: ipv4net
               esp-group: text
               allow-nat-networks: [enable|disable]
               allow-public-networks: [enable|disable]
            }
         }
      }
      rekey-timers {
         rekey-time: 10-86400
         rekey-random: 0-100
      }
      logging {
         facility: [daemon|local0..local7]
         level: [emerg|crit|err|warning|alert|notice|info|debug]
         log-modes [all|raw|crypt|parsing|emitting|control|
            private] {}
      }
   }
   rsa-keys {
      local-key {
         file: file-name
      rsa-key-name text {
         rsa-key: key-data
   }
}
```

Glossary

AS	*See* Autonomous System.
Autonomous System	A routing domain that is under one administrative authority, and which implements its own routing policies. A key concept in BGP.
BGP	Border Gateway Protocol.
Bootstrap Router	A PIM-SM router that chooses the RPs for a domain from amongst a set of candidate RPs.
BSR	*See* Bootstrap Router.
Candidate RP	A PIM-SM router that is configured to be a candidate to be an RP. The Bootstrap Router will then choose the RPs from the set of candidates.
Dynamic Route	A route learned from another router via a routing protocol such as RIP or BGP.
EGP	*See* Exterior Gateway Protocol.
Exterior Gateway Protocol	A routing protocol used to route between Autonomous Systems. The main example is BGP.
IGMP	Internet Group Management Protocol. *TBD*
IGP	*See* Interior Gateway Protocol.
Interior Gateway Protocol	A routing protocol used to route within an Autonomous System. Examples include RIP, OSPF and IS-IS.
MLD	Multicast Listener Discovery protocol. TBD
MRIB	*See* Multicast RIB.

Multicast RIB	The part of the RIB that holds multicast routes. These are not directly used for forwarding, but instead are used by multicast routing protocols such as PIM-SM to perform RPF checks when building the multicast distribution tree.
OSPF	Open Shortest Path First. An IGP routing protocol based on a link-state algorithm. Used to route within medium to large networks.
PIM-SM	Protocol Independent Multicast, Sparse-Mode TBD
Rendezvous Point	A router used in PIM-SM as part of the rendezvous process by which new senders are grafted on to the multicast tree.
Reverse Path Forwarding	Many multicast routing protocols such as PIM-SM build a multicast distribution tree based on the best route back from each receiver to the source, hence multicast packets will be forwarded along the reverse of the path to the source.
RIB	*See* Routing Information Base.
RIP	Routing Information Protocol. *TBD*
Routing Information Base	The collection of routes learned from all the dynamic routing protocols running on the router. Subdivided into a Unicast RIB for unicast routes and a Multicast RIB.
RP	*See* Rendezvous Point.
RPF	*See* Reverse Path Forwarding.
Static Route	A route that has been manually configured on the router.
xorpsh	XORP command shell.
xorp rtrmgr	XORP router manager process.