JUNIPER NETWORKS

# Junos® OS

## Routing Protocols Overview

Release
12.3

Published: 2012-12-08

Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net



*Junos® OS Routing Protocols Overview*
12.3


The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

## END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at http://www.juniper.net/support/eula.html. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

# Table of Contents

# List of Figures

# List of Tables

# About the Documentation



## Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at http://www.juniper.net/techpubs/.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at http://www.juniper.net/books.

## Supported Platforms

For the features described in this document, the following platforms are supported:

- T Series
- MX Series
- M Series
- J Series
- SRX Series

## Using the Examples in This Manual

If you want to use the examples in this manual, you can use the **load merge** or the **load merge relative** command. These commands cause the software to merge the incoming

configuration into the current candidate configuration. The example does not become active until you commit the candidate configuration.

If the example configuration contains the top level of the hierarchy (or multiple hierarchies), the example is a *full example*. In this case, use the **load merge** command.

If the example configuration does not start at the top level of the hierarchy, the example is a *snippet*. In this case, use the **load merge relative** command. These procedures are described in the following sections.

## Merging a Full Example

To merge a full example, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration example into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following configuration to a file and name the file **ex-script.conf**. Copy the **ex-script.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   system {
     scripts {
       commit {
         file ex-script.xsl;
       }
     }
   }
   interfaces {
     fxp0 {
       disable;
       unit 0 {
         family inet {
           address 10.0.0.1/24;
         }
       }
     }
   }
   ```

2. Merge the contents of the file into your routing platform configuration by issuing the **load merge** configuration mode command:

   ```
   [edit]
   user@host# load merge /var/tmp/ex-script.conf
   load complete
   ```

## Merging a Snippet

To merge a snippet, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration snippet into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following snippet to a file and name the file **ex-script-snippet.conf**. Copy the **ex-script-snippet.conf** file to the **/var/tmp** directory on your routing platform.

```
commit {
  file ex-script-snippet.xsl; }
```

2. Move to the hierarchy level that is relevant for this snippet by issuing the following configuration mode command:

```
[edit]
user@host# edit system scripts
[edit system scripts]
```

3. Merge the contents of the file into your routing platform configuration by issuing the **load merge relative** configuration mode command:

```
[edit system scripts]
user@host# load merge relative /var/tmp/ex-script-snippet.conf
load complete
```

For more information about the **load** command, see the CLI User Guide.

## Documentation Conventions

Table 1 on page xi defines notice icons used in this guide.

### Table 1: Notice Icons

| Icon | Meaning | Description |
|---|---|---|
|  | Informational note | Indicates important features or instructions. |
|  | Caution | Indicates a situation that might result in loss of data or hardware damage. |
|  | Warning | Alerts you to the risk of personal injury or death. |
|  | Laser warning | Alerts you to the risk of personal injury from a laser. |

Table 2 on page xi defines the text and syntax conventions used in this guide.

### Table 2: Text and Syntax Conventions

| Convention | Description | Examples |
|---|---|---|
| **Bold text like this** | Represents text that you type. | To enter configuration mode, type the**configure** command:<br><br>user@host> **configure** |
| `Fixed-width text like this` | Represents output that appears on the terminal screen. | `user@host>` **show chassis alarms**<br><br>`No alarms currently active` |

Table 2: Text and Syntax Conventions *(continued)*

| Convention | Description | Examples |
|---|---|---|
| *Italic text like this* | • Introduces or emphasizes important new terms.<br>• Identifies book names.<br>• Identifies RFC and Internet draft titles. | • A policy *term* is a named structure that defines match conditions and actions.<br>• *Junos OS System Basics Configuration Guide*<br>• RFC 1997, *BGP Communities Attribute* |
| *Italic text like this* | Represents variables (options for which you substitute a value) in commands or configuration statements. | Configure the machine's domain name:<br>[edit]<br>root@# **set system domain-name** *domain-name* |
| **Text like this** | Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components. | • To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level.<br>• The console port is labeled **CONSOLE**. |
| < > (angle brackets) | Enclose optional keywords or variables. | **stub <default-metric** ***metric*****>;** |
| \| (pipe symbol) | Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity. | **broadcast \| multicast**<br>**(*string1* \| *string2* \| *string3*)** |
| # (pound sign) | Indicates a comment specified on the same line as the configuration statement to which it applies. | **rsvp { # Required for dynamic MPLS only** |
| [ ] (square brackets) | Enclose a variable for which you can substitute one or more values. | **community name members [** ***community-ids*** **]** |
| Indention and braces ( { } ) | Identify a level in the configuration hierarchy. | [edit]<br>routing-options {<br>  static {<br>  route default {<br>  nexthop *address*;<br>  retain;<br>  }<br>  }<br>} |
| ; (semicolon) | Identifies a leaf statement at a configuration hierarchy level. | |
| J-Web GUI Conventions | | |
| **Bold text like this** | Represents J-Web graphical user interface (GUI) items you click or select. | • In the Logical Interfaces box, select **All Interfaces**.<br>• To cancel the configuration, click **Cancel**. |
| **>** (bold right angle bracket) | Separates levels in a hierarchy of J-Web selections. | In the configuration editor hierarchy, select **Protocols>Ospf**. |

## Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can send your comments to techpubs-comments@juniper.net, or fill out the documentation feedback form at https://www.juniper.net/cgi-bin/docbugreport/ . If you are using e-mail, be sure to include the following information with your comments:

- Document or topic name
- URL or page number
- Software release version (if applicable)

## Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or JNASC support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at http://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf.
- Product warranties—For product warranty information, visit http://www.juniper.net/support/warranty/.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

### Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: http://www.juniper.net/customers/support/
- Search for known bugs: http://www2.juniper.net/kb/
- Find product documentation: http://www.juniper.net/techpubs/
- Find solutions and answer questions using our Knowledge Base: http://kb.juniper.net/
- Download the latest versions of software and review release notes: http://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: https://www.juniper.net/alerts/

- Join and participate in the Juniper Networks Community Forum: http://www.juniper.net/company/communities/
- Open a case online in the CSC Case Management tool: http://www.juniper.net/cm/

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://tools.juniper.net/SerialNumberEntitlementSearch/

## Opening a Case with JTAC

You can open a case with JTAC on the Web or by telephone.

- Use the Case Management tool in the CSC at http://www.juniper.net/cm/.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see http://www.juniper.net/support/requesting-support.html.

PART 1

# Overview

CHAPTER 1

# Introduction to Routing Protocols



## Routing Databases Overview

To use the routing capabilities of a Juniper Networks device, you must understand the fundamentals of IP routing and the routing protocols that are primarily responsible for the transmission of unicast traffic. To understand this topic, you need a basic understanding of IP addressing and TCP/IP.

The Junos® operating system (Junos OS) maintains two databases for routing information:

- Routing table—Contains all the routing information learned by all routing protocols.
- Forwarding table—Contains the routes actually used to forward packets through the router.

In addition, the interior gateway protocols (IGPs), IS-IS, and OSPF maintain link-state databases.

This section includes the following topics:



### Routing Protocol Databases

Each IGP routing protocol maintains a database of the routing information it has learned from other routers running the same protocol and uses this information as defined and

required by the protocol. Routing information that is shared within an AS is transmitted by an interior gateway protocol (IGP).

Of the different IGPs, the most common are RIP, OSPF, and IS-IS. IS-IS and OSPF use the routing information they received to maintain link-state databases, which they use to determine which adjacent neighbors are operational and to construct network topology maps. IGPs are designed to be fast acting and light duty. They typically incorporate only a moderate security system, because trusted internal peers do not require the stringent security measures that untrusted peers require. As a result, you can usually begin routing within an AS by enabling the IGP on all internal interfaces and performing minimal additional configuration. You do not need to establish individual adjacencies.

IS-IS and OSPF use the Dijkstra algorithm, and RIP and RIPng use the Bellman-Ford algorithm to determine the best route or routes (if there are multiple equal-cost routes) to reach each destination and install these routes into the Junos OS routing table.

Routing information that is shared with a peer AS is transmitted by an exterior gateway protocol (EGP). The primary EGP in use in almost all networks is the Border Gateway Protocol (BGP). BGP is designed to be very secure. Individual connections must be explicitly configured on each side of the link. As a result, although large numbers of connections are difficult to configure and maintain, each connection is secure.

When you configure a protocol on an interface, you must also configure a protocol family on that interface.

## Junos Routing Tables

The Junos OS routing table is used by the routing protocol process to maintain its database of routing information. In this table, the routing protocol process stores statically configured routes, directly connected interfaces (also called *direct routes* or *interface routes*), and all routing information learned from all routing protocols. The routing protocol process uses this collected routing information to select the *active route* to each destination, which is the route that actually is used to forward packets to that destination. To route traffic from a source host to a destination host, the devices through which the traffic will pass must learn the path that the packet is to take. Once learned, the information is stored in routing tables. The routing table maintains a list of all the possible paths from point A to point B.

By default, the Junos OS maintains three routing tables: one for unicast routes, another for multicast routes, and a third for MPLS. You can configure additional routing tables to support situations where you need to separate a particular group of routes or where you need greater flexibility in manipulating routing information. In general, most operations can be performed without resorting to the complexity of additional routing tables. However, creating additional routing tables has several specific uses, including importing interface routes into more than one routing table, applying different routing policies when exporting the same route to different peers, and providing greater flexibility with incongruent multicast topologies.

Each routing table is identified by a name, which consists of the protocol family followed by a period and a small, nonnegative integer. The protocol family can be **inet** (Internet),

**iso** (ISO), or **mpls** (MPLS). The following names are reserved for the default routing tables maintained by the Junos OS:

- **inet.0**—Default IP version 4 (IPv4) unicast routing table
- **inet6.0**—Default IP version 6 (IPv6) unicast routing table
- **instance-name.inet.0**—Unicast routing table for a particular routing instance
- **inet.1**—Multicast forwarding cache
- **inet.2**—Unicast routes used for multicast reverse path forwarding (RPF) lookup
- **inet.3**—MPLS routing table for path information
- **mpls.0**—MPLS routing table for label-switched path (LSP) next hops

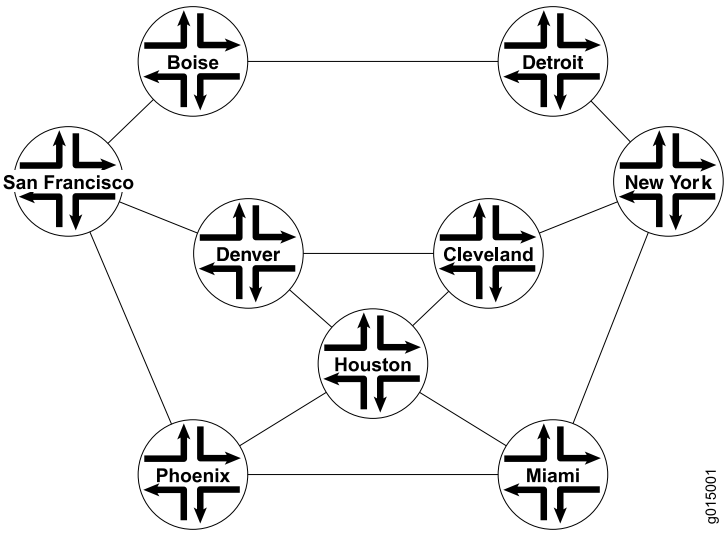
NOTE: For clarity, this topic contains general discussions of routing tables as if there were only one table. However, when it is necessary to distinguish among the routing tables, their names are explicitly used.

## Networks and Subnetworks

Large groups of machines that are interconnected and can communicate with one another form networks. Typically, networks identify large systems of computers and devices that are owned or operated by a single entity. Traffic is routed between or through the networks as data is passed from host to host.

Figure 1 on page 5 shows a simple network of routers.

### Figure 1: Simple Network Topology




This simple network provides multiple ways to get from host San Francisco to host Miami. The packet can follow the path through Denver and Cleveland. Alternatively, the packet can be routed through Phoenix and directly to Miami. The routing table includes all the

possible paths and combinations—an exhaustive list of all the ways to get from the source to the destination.

The routing table must include every possible path from a source to a destination. Routing tables for the network in Figure 1 on page 5 must include entries for San Francisco-Denver, San Francisco-Cleveland, San Francisco-Miami, Denver-Cleveland, and so on. As the number of sources and destinations increases, the routing table quickly becomes large. The unwieldy size of routing tables is the primary reason for the division of networks into subnetworks.

As networks grow large, the ability to maintain the network and effectively route traffic between hosts within the network becomes increasingly difficult. To accommodate growth, networks are divided into subnetworks. Fundamentally, subnetworks behave exactly like networks, except that they are identified by a more specific network address and subnet mask (destination prefix). Subnetworks have routing gateways and share routing information in exactly the same way as large networks.

## Forwarding Tables

Routing is the transmission of data packets from a source to a destination address. It involves delivering a message across a network or networks. This process has two primary components: the exchange of routing information to forward packets accurately from source to destination and the packet-forwarding procedure.

For packets to be correctly forwarded to the appropriate host address, the host must have a unique numeric identifier or IP address. The unique IP address of the destination host forms entries in the routing table. These entries are primarily responsible for determining the path that a packet traverses when transmitted from source to destination.

The Junos OS installs all active routes from the routing table into the forwarding table. The active routes are used to forward packets to their destinations.

The Junos OS kernel maintains a master copy of the forwarding table. It copies the forwarding table to the Packet Forwarding Engine, which is the part of the router responsible for forwarding packets.

If the routing table is a list of all the possible paths a packet can take, the forwarding table is a list of only the best routes to a particular destination. The best path is determined according to the particular routing protocol being used, but generally the number of hops between the source and destination determines the best possible route.

In the network shown in Figure 1 on page 5, because the path with the fewest number of hops from San Francisco to Miami is through Phoenix, the forwarding table distills all the possible San Francisco-Miami routes into the single route through Phoenix. All traffic with a destination address of Miami is sent directly to the next hop, Phoenix.

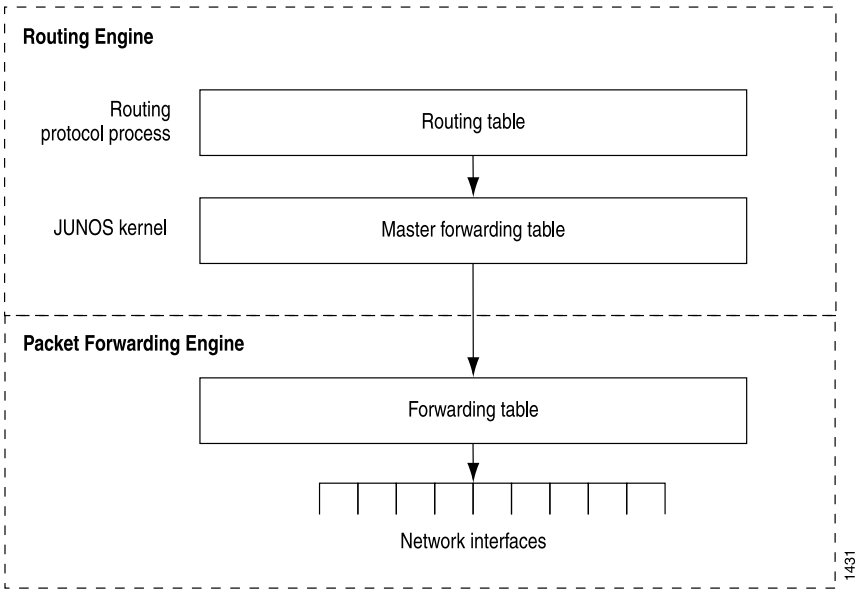
After it receives a packet, the Phoenix router performs another route lookup, using the same destination address. The Phoenix router then routes the packet appropriately. Although it considers the entire path, the router at any individual hop along the way is responsible only for transmitting the packet to the next hop in the path. If the Phoenix router is managing its traffic in a particular way, it might send the packet through Houston on its route to Miami. This scenario is likely if specific customer traffic is treated as priority

traffic and routed through a faster or more direct route, while all other traffic is treated as nonpriority traffic.

## How the Routing and Forwarding Tables Are Synchronized

The Junos OS routing protocol process is responsible for synchronizing the routing information between the routing and forwarding tables. To do this, the routing protocol process calculates the active routes from all the routes in the routing table and installs them into the forwarding table. The routing protocol process then copies the forwarding table to the router's Packet Forwarding Engine, the part of the router that forwards packets. Figure 2 on page 7 illustrates how the routing tables are synchronized.

### Figure 2: Synchronizing Routing Exchange Between the Routing and Forwarding Tables




## NetFlow V9 Support

NetFlow Services Export Version 9 (NetFlow V9) provides an extensible and flexible method for using templates to observe packets on a router. Each template indicates the format in which the router exports data.

NetFlow V9 is supported in Junos OS in the adaptive service PIC module.

This feature supports Netflow V5 or V8 for flow-based devices.

## Route Preferences Overview

For unicast routes, the Junos OS routing protocol process uses the information in its routing table, along with the properties set in the configuration file, to choose an *active route* for each destination. While the Junos OS might know of many routes to a destination, the active route is the preferred route to that destination and is the one that is installed in the forwarding table and used when actually routing packets.

The routing protocol process generally determines the active route by selecting the route with the lowest preference value. The preference value is an arbitrary value in the range from 0 through 4,294,967,295 ($2^{32} - 1$) that the software uses to rank routes received from different protocols, interfaces, or remote systems.

The preference value is used to select routes to destinations in external autonomous systems (ASs) or routing domains; it has no effect on the selection of routes within an AS (that is, within an interior gateway protocol [IGP]). Routes within an AS are selected by the IGP and are based on that protocol's metric or cost value.

This section includes the following topics:



### Autonomous Systems

A large network or collection of routers under a single administrative authority is termed an *autonomous system* (AS). Autonomous systems are identified by a unique numeric identifier that is assigned by the Internet Assigned Numbers Authority (IANA). Typically, the hosts within an AS are treated as internal peers, and hosts in a peer AS are treated as external peers. The status of the relationship between hosts—internal or external—governs the protocol used to exchange routing information.

### Alternate and Tiebreaker Preferences

The Junos OS provides support for alternate and tiebreaker preferences, and some of the routing protocols, including BGP and label switching, use these additional preferences. With these protocols, you can specify a primary route preference (by including the **preference** statement in the configuration), and a secondary preference that is used as a tiebreaker (by including the **preference2** statement). You can also mark route preferences with additional route tiebreaker information by specifying a color and a tiebreaker color (by including the **color** and **color2** statements in the configuration).

The software uses a 4-byte value to represent the route preference value. When using the preference value to select an active route, the software first compares the primary route preference values, choosing the route with the lowest value. If there is a tie and a

secondary preference has been configured, the software compares the secondary preference values, choosing the route with the lowest value. The secondary preference values must be included in a set for the preference values to be considered.

## Multiple Active Routes

The IGPs compute equal-cost multipath next hops, and IBGP picks up these next hops. When there are multiple, equal-cost next hops associated with a route, the routing protocol process installs only one of the next hops in the forwarding path with each route, randomly selecting which next hop to install. For example, if there are 3 equal-cost paths to an exit routing device and 900 routes leaving through that routing device, each path ends up with about 300 routes pointing at it. This mechanism provides load distribution among the paths while maintaining packet ordering per destination.

BGP multipath does not apply to paths that share the same MED-plus-IGP cost yet differ in IGP cost. Multipath path selection is based on the IGP cost metric, even if two paths have the same MED-plus-IGP cost.

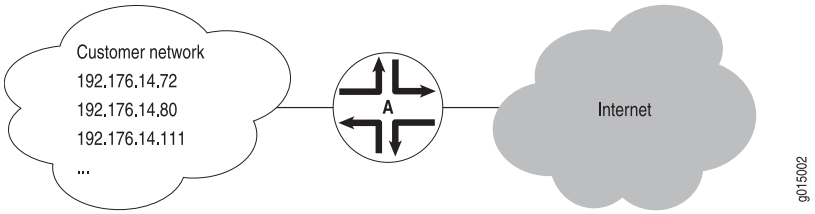
## Dynamic and Static Routing

Entries are imported into a router's routing table from dynamic routing protocols or by manual inclusion as static routes. Dynamic routing protocols allow routers to learn the network topology from the network. The routers within the network send out routing information in the form of route advertisements. These advertisements establish and communicate active destinations, which are then shared with other routers in the network.

Although dynamic routing protocols are extremely useful, they have associated costs. Because they use the network to advertise routes, dynamic routing protocols consume bandwidth. Additionally, because they rely on the transmission and receipt of route advertisements to build a routing table, dynamic routing protocols create a delay (latency) between the time a router is powered on and the time during which routes are imported into the routing table. Some routes are therefore effectively unavailable until the routing table is completely updated, when the router first comes online or when routes change within the network (due to a host going offline, for example).

Static routing avoids the bandwidth cost and route import latency of dynamic routing. Static routes are manually included in the routing table, and never change unless you explicitly update them. Static routes are automatically imported into the routing table when a router first comes online. Additionally, all traffic destined for a static address is routed through the same router. This feature is particularly useful for networks with customers whose traffic must always flow through the same routers. Figure 3 on page 9 shows a network that uses static routes.

### Figure 3: Static Routing Example



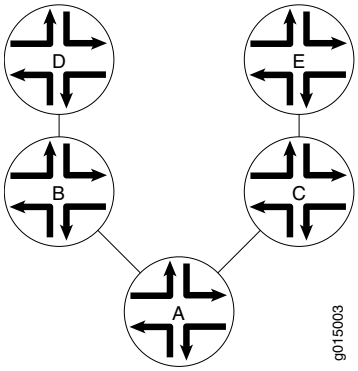
In Figure 3 on page 9, the customer routes in the **192.176.14/24** subnetwork are static routes. These are hard links to specific customer hosts that never change. Because all traffic destined for any of these routes is forwarded through Router A, these routes are included as static routes in Router A's routing table. Router A then advertises these routes to other hosts so that traffic can be routed to and from them.

## Route Advertisements

The routing table and forwarding table contain the routes for the routers within a network. These routes are learned through the exchange of route advertisements. Route advertisements are exchanged according to the particular protocol being employed within the network.

Generally, a router transmits hello packets out each of its interfaces. Neighboring routers detect these packets and establish adjacencies with the router. The adjacencies are then shared with other neighboring routers, which allows the routers to build up the entire network topology in a topology database, as shown in Figure 4 on page 10.

### Figure 4: Route Advertisement




In Figure 4 on page 10, Router A sends out hello packets to each of its neighbors. Routers B and C detect these packets and establish an adjacent relationship with Router A. Router B and C then share this information with their neighbors, Routers D and E, respectively. By sharing information throughout the network, the routers create a network topology, which they use to determine the paths to all possible destinations within the network. The routes are then distilled into the forwarding table of best routes according to the route selection criteria of the protocol in use.

## Route Aggregation

As the number of hosts in a network increases, the routing and forwarding tables must establish and maintain more routes. As these tables become larger, the time routers require to look up particular routes so that packets can be forwarded becomes prohibitive. The solution to the problem of growing routing tables is to group (aggregate) the routers by subnetwork, as shown in Figure 5 on page 11.

Figure 5: Route Aggregation

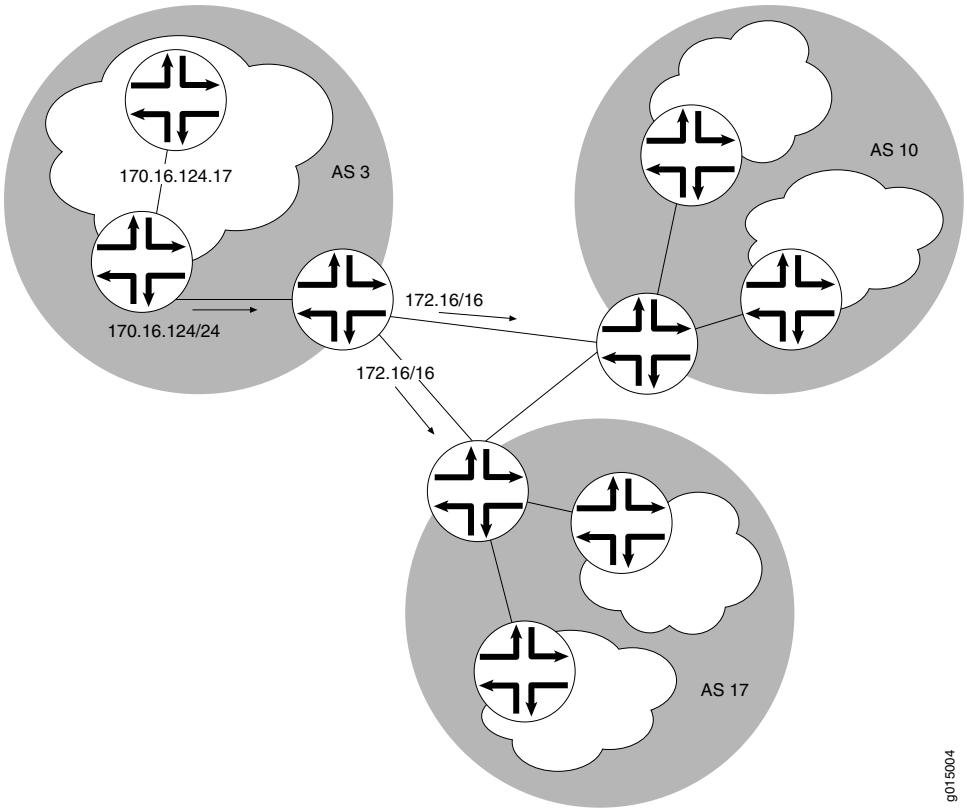



Figure 5 on page 11 shows three different ASs. Each AS contains multiple subnetworks with thousands of host addresses. To allow traffic to be sent from any host to any host, the routing tables for each host must include a route for each destination. For the routing tables to include every combination of hosts, the flooding of route advertisements for each possible route becomes prohibitive. In a network of hosts numbering in the thousands or even millions, simple route advertisement is not only impractical but impossible.

By employing route aggregation, instead of advertising a route for each host in AS 3, the gateway router advertises only a single route that includes all the routes to all the hosts within the AS. For example, instead of advertising the particular route **170.16.124.17**, the AS 3 gateway router advertises only **170.16/16**. This single route advertisement encompasses all the hosts within the **170.16/16** subnetwork, which reduces the number of routes in the routing table from $2^{16}$ (one for every possible IP address within the subnetwork) to 1. Any traffic destined for a host within the AS is forwarded to the gateway router, which is then responsible for forwarding the packet to the appropriate host.

Similarly, in this example, the gateway router is responsible for maintaining $2^{16}$ routes within the AS (in addition to any external routes). The division of this AS into subnetworks allows for further route aggregation to reduce this number. In the subnetwork in the example, the subnetwork gateway router advertises only a single route (**170.16.124/24**), which reduces the number of routes from $2^{8}$ to 1.

## Routing Instances Overview

You can create multiple instances of BGP, IS-IS, LDP, Multicast Source Discovery Protocol (MSDP), OSPF version 2 (usually referred to simply as OSPF), OSPF version 3 (OSPFv3), Protocol Independent Multicast (PIM), RIP, RIP next generation (RIPng), and static routes by including statements at the following hierarchy levels:

- **[edit routing-instances *routing-instance-name* protocols]**
- **[edit logical-systems *logical-system-name* routing-instances *routing-instance-name* protocols]**

Only one instance of each protocol can be configured in a single routing instance.

NOTE: You can also create multiple routing instances for separating routing tables, routing policies, and interfaces for individual DHCP wholesale subscribers (retailers) in a layer 3 wholesale network. For information about how to configure layer 3 wholesale network services, see the Junos OS Subscriber Management, Release 12.3.

A routing instance is a collection of routing tables, interfaces, and routing protocol parameters. The set of interfaces belongs to the routing tables, and the routing protocol parameters control the information in the routing tables. There can be multiple routing tables for a single routing instance—for example, unicast IPv4, unicast IPv6, and multicast IPv4 routing tables can exist in a single routing instance. Routing protocol parameters and options control the information in the routing tables.

You can configure eight types of routing instances: forwarding, Layer 2 control (MX Series routers only), Layer 2 virtual private network (VPN), nonforwarding, VPN routing and forwarding (VRF), virtual router, virtual private LAN service (VPLS), and virtual switch (MX Series routers only).

Each routing instance has a unique name and a corresponding IP unicast table. For example, if you configure a routing instance with the name **my-instance**, the corresponding IP unicast table is **my-instance.inet.0**. All routes for **my-instance** are installed into **my-instance.inet.0**.

NOTE: The default routing instance, master, refers to the main inet.0 routing table. The master routing instance is reserved and cannot be specified as a routing instance.

Each routing instance consists of sets of the following:

- Routing tables
- Interfaces that belong to these routing tables (optional, depending upon the routing instance type)

- Routing option configurations

You can configure eight types of routing instances:

- Forwarding—Use this routing instance type for filter-based forwarding applications. For this instance type, there is no one-to-one mapping between an interface and a routing instance. All interfaces belong to the default instance inet.0.
- Layer 2 Backhaul VPN—(MX Series routers only) Use this routing instance type to provide support for Layer 2 wholesale VLAN packets with no existing corresponding logical interface. When using this instance, the router learns both the outer tag and inner tag of the incoming packets, when the **instance-role** statement is defined as **access**, or the outer VLAN tag only, when the **instance-role** statement is defined as **nni**.
- Layer2-control—(MX Series routers only) Use this routing instance type for RSTP or MSTP in customer edge interfaces of a VPLS routing instance. This instance type cannot be used if the customer edge interface is multihomed to two provider edge interfaces. If the customer edge interface is multihomed to two provider edge interfaces, use the default BPDU tunneling.
- Layer 2 VPN—Use this routing instance type for Layer 2 virtual private network (VPN) implementations.
- Nonforwarding—Use this routing instance type when a separation of routing table information is required. There is no corresponding forwarding table. All routes are installed into the default forwarding table. IS-IS instances are strictly nonforwarding instance types.
- Virtual router—Similar to a VPN routing and forwarding instance type, but used for non-VPN-related applications. There are no virtual routing and forwarding (VRF) import, VRF export, VRF target, or route distinguisher requirements for this instance type.
- Virtual switch—(MX Series routers only) Use the virtual switch instance type to isolate a LAN segment with its Spanning Tree Protocol (STP) instance and separates its VLAN identifier space. For more detail information about configuring a virtual switch, see the Junos OS Layer 2 Configuration Guide and the JUNOS® MX Series 3D Universal Edge Routers Solutions, Release 12.3.
- VPLS—Use the virtual private local-area network service (VPLS) routing instance type for point-to-multipoint LAN implementations between a set of sites in a VPN.
- VRF—Use the VPN routing and forwarding routing (VRF) instance type for Layer 3 VPN implementations. This routing instance type has a VPN routing table as well as a corresponding VPN forwarding table. For this instance type, there is a one-to-one mapping between an interface and a routing instance. Each VRF instance corresponds with a forwarding table. Routes on an interface go into the corresponding forwarding table.

Configure global routing options and protocols for the master instance by including statements at the **[edit protocols]** and **[edit routing-options]** hierarchy levels. Routes are installed into the master routing instance **inet.0** by default, unless a routing instance is specified.

Multiple instances of BGP, OSPF, and RIP are used for Layer 3 VPN implementation. The multiple instances of BGP, OSPF, and RIP keep routing information for different VPNs separate. The VRF instance advertises routes from the customer edge (CE) router to the provider edge (PE) router and advertises routes from the PE router to the CE router. Each VPN receives only routing information belonging to that VPN.

Forwarding instances are used to implement filter-based forwarding for Common Access Layer applications.

PIM instances are used to implement multicast over VPN applications.

Nonforwarding instances of IS-IS and OSPF can be used to separate a very large network into smaller administrative entities. Instead of configuring a large number of filters, nonforwarding instances can be used to filter routes, thereby instantiating policy. Nonforwarding instances can be used to reduce the amount of routing information advertised throughout all components of a network. Routing information associated with a particular instance can be announced where required, instead of being advertised to the whole network.

Layer 2 VPN instances are used for Layer 2 VPN implementation.

Virtual router instances are similar to a VPN routing and forwarding instance type, but used for non-VPN-related applications. There are no VRF import, VRF export, VRF target, or route distinguisher requirements for this instance type.

Use the VPLS routing instance type for point-to-multipoint LAN implementations between a set of sites in a VPN.

To configure a routing instance type, use the **instance-type** statement at the [**edit routing-instances** ***routing-instance-name***] hierarchy level.

To configure a routing instance, specify the following parameters:

- Name of the routing instance. Each routing instance has a unique name and a corresponding IP unicast table. For example, if you configure a routing instance with the name my-instance, its corresponding IP unicast table is my-instance.inet.0. All routes for my-instance are installed into **my-instance.inet.0**.

NOTE: **You cannot specify a routing-instance name of *default* or include special characters within the name of a routing instance.**

- Type of routing instance.
- The interfaces that are bound to the routing instance. Interfaces not required for the forwarding routing instance type.

To configure a routing instance, use the **routing-instances** statement at the [**edit**] hierarchy level.

You can create an instance of BGP, IS-IS, OSPF, OSPFv3, RIP, or RIPng by including configuration statements at the **[edit routing-instances *routing-instance-name* protocols]** hierarchy level. You can also configure static routes for the routing instance.

**Related Documentation**

- Junos OS VPNs Configuration Guide
- Junos OS Layer 2 Configuration Guide
- JUNOS® MX Series 3D Universal Edge Routers Solutions, Release 12.3

## Equal-Cost Paths and Load Sharing

For equal-cost paths, load sharing is based on the BGP next hop. For example, if four prefixes all point to a next hop and there is more than one equal-cost path to that next hop, the routing protocol process uses a hash algorithm to choose the path among the four prefixes. Also, for each prefix, the routing protocol process installs a single forwarding entry pointing along one of the paths. The routing software does not rehash the path taken as prefixes pointing to the next hop come and go, but it does rehash if the number of paths to the next hop changes. Because a prefix is tied to a particular path, packet reordering should not happen. The degree of load sharing improves as the number of prefixes increases.

## IPv6 Overview

IP version 6 (IPv6) is the latest version of IP. IP enables numerous nodes on different networks to interoperate seamlessly. IP version 4 (IPv4) is currently used in intranets and private networks, as well as the Internet. IPv6 is the successor to IPv4, and is based for the most part on IPv4.

IPv4 has been widely deployed and used to network the Internet today. With the rapid growth of the Internet, enhancements to IPv4 are needed to support the influx of new subscribers, Internet-enabled devices, and applications. IPv6 is designed to enable the global expansion of the Internet.

IPv6 builds upon the functionality of IPv4, providing improvements to addressing, configuration and maintenance, and security.

IPv6 offers the following benefits:

- Expanded addressing capabilities—IPv6 provides a larger address space. IPv6 addresses consist of 128 bits, while IPv4 addresses consist of 32 bits. 128-bit addressing increases the address space by approximately 1029 unique addresses, enough to last for the forseeable future.
- Header format simplification—IPv6 packet header format is designed to be efficient. IPv6 standardizes the size of the packet header to 40 bytes, divided into 8 fields.
- Improved support for extensions and options—Extension headers carry Internet-layer information and have a standard size and structure.
- Flow labeling capability—Flow labels provide consistent handling of packets belonging to the same flow.
- Improved privacy and security—IPv6 supports extensions for authentication and data integrity, which enhance privacy and security.

NOTE: **When configuring IPv6 addressing and routing on a J Series device, you must enable IPv6 in secure context.**

This section discusses the following topics:


- IPv6 Packet Headers on page 16
- IPv6 Addressing on page 17


### IPv6 Packet Headers

IPv6 headers are different from IPv4 headers.

This section discusses the following topics that provide background information about IPv6 headers:



### Header Structure

IPv6 packet headers contain many of the fields found in IPv4 packet headers; some of these fields have been modified from IPv4. The 40-byte IPv6 header consists of the following 8 fields:

- Traffic class—Class-of-service (CoS) priority of the packet. Previously the type-of-service (ToS) field in IPv4. However, the semantics of this field (for example, DiffServ code points) are identical to IPv4.
- Destination address—Final destination node address for the packet.
- Flow label—Packet flows requiring a specific class of service. The flow label identifies all packets belonging to a specific flow, and routers can identify these packets and handle them in a similar fashion.
- Hop limit—Maximum number of hops allowed. Previously the time-to-live (TTL) field in IPv4.
- Next header—Next extension header to examine. Previously the protocol field in IPv4.
- Payload length—Length of the IPv6 payload. Previously the total length field in IPv4.
- Source address—Address of the source node sending the packet.
- Version—Version of IP.

### Extension Headers

In IPv6, *extension headers* are used to encode optional Internet-layer information.

Extension headers are placed between the IPv6 header and the upper layer header in a packet.

Extension headers are chained together using the next header field in the IPv6 header. The next header field indicates to the router which extension header to expect next. If there are no more extension headers, the next header field indicates the upper layer header (TCP header, User Datagram Protocol [UDP] header, ICMPv6 header, an encapsulated IP packet, or other items).

## IPv6 Addressing

IPv6 uses a 128-bit addressing model. This creates a much larger address space than IPv4 addresses, which are made up of 32 bits. IPv6 addresses also contain a scope field that categorizes what types of applications are suitable for the address. IPv6 does not support broadcast addresses, but instead uses multicast addresses to serve this role. In addition, IPv6 also defines a new type of address called *anycast*.

NOTE: You cannot configure a subnet zero IPv6 address because RFC 2461 reserves the subnet-zero address for anycast addresses, and Junos OS complies with the RFC.

This section discusses the following topics that provide background information about IPv6 addressing:



## Address Representation

IPv6 addresses consist of 8 groups of 16-bit hexadecimal values separated by colons (:). The IPv6 address format is as follows:

*aaaa*:*aaaa*:*aaaa*:*aaaa*:*aaaa*:*aaaa*:*aaaa*:*aaaa*

***aaaa*** is a 16-bit hexadecimal value, and ***a*** is a 4-bit hexadecimal value. Following is an example of an actual IPv6 address:

3FFE:0000:0000:0001:0200:F8FF:FE75:50DF

You can omit the leading zeros, as shown:

3FFE:0:0:1:200:F8FF:FE75:50DF

You can compress 16-bit groups of zeros to the notation :: (two colons), as shown here, but only once per address:

3FFE::1:200:F8FF:FE75:50DF

## Address Types

There are three types of IPv6 addresses:

- Unicast—For a single interface.
- Multicast—For a set of interfaces on the same physical medium. A packet is sent to all of the interfaces associated with the address.
- Anycast—For a set of interfaces on different physical mediums. A packet is sent to only one of the interfaces associated with this address, not to all the interfaces.

## Address Scope

IPv6 addresses have *scope*, which identifies the application suitable for the address. Unicast and multicast addresses support scoping.

Unicast addresses support two types of scope: *global* scope and *local* scope. There are two types of local scope: *link-local* addresses and *site-local* addresses. Link-local unicast addresses are used within a single network link. The first ten bits of the prefix identify the

address as a link-local address. Link-local addresses cannot be used outside a network link. Site-local unicast addresses are used within a site or intranet. A site consists of multiple network links, and site-local addresses identify nodes inside the intranet. Site-local addresses cannot be used outside the site.

Multicast addresses support 16 different types of scope, including node, link, site, organization, and global scope. A 4-bit field in the prefix identifies the scope.

### Address Structure

Unicast addresses identify a single interface. The address consists of *n* bits for the prefix, and 128 – *n* bits for the interface ID.

Multicast addresses identify a set of interfaces. The address is made up of the first 8 bits of all ones, a 4-bit flags field, a 4-bit scope field, and a 112-bit group ID:

11111111 | *flags* | *scope* | *group ID*

The first octet of ones identifies the address as a multicast address. The flags field identifies whether the multicast address is a well-known address or a transient multicast address. The scope field identifies the scope of the multicast address. The 112-bit group ID identifies the multicast group.

Similar to multicast addresses, anycast addresses identify a set of interfaces. However, packets are sent to only one of the interfaces, not to all interfaces. Anycast addresses are allocated from the normal unicast address space and cannot be distinguished from a unicast address in format.

PART 2

# Administration

CHAPTER 2

# Routing Protocols Reference



## Understanding BGP Path Selection

For each prefix in the routing table, the routing protocol process selects a single best path. After the best path is selected, the route is installed in the routing table. The best path becomes the active route if the same prefix is not learned by a protocol with a lower (more preferred) global preference value, also known as the administrative distance. The algorithm for determining the active route is as follows:

1. Verify that the next hop can be resolved.
2. Choose the path with the lowest preference value (routing protocol process preference).

   Routes that are not eligible to be used for forwarding (for example, because they were rejected by routing policy or because a next hop is inaccessible) have a preference of –1 and are never chosen.
3. Prefer the path with higher local preference.

   For non-BGP paths, choose the path with the lowest **preference2** value.
4. If the accumulated interior gateway protocol (AIGP) attribute is enabled, prefer the path with the lower AIGP attribute.
5. Prefer the path with the shortest autonomous system (AS) path value (skipped if the **as-path-ignore** statement is configured).

   A confederation segment (sequence or set) has a path length of 0. An AS set has a path length of 1.
6. Prefer the route with the lower origin code.

   Routes learned from an IGP have a lower origin code than those learned from an exterior gateway protocol (EGP), and both have lower origin codes than incomplete routes (routes whose origin is unknown).
7. Prefer the path with the lowest multiple exit discriminator (MED) metric.

Depending on whether nondeterministic routing table path selection behavior is configured, there are two possible cases:

- If nondeterministic routing table path selection behavior is not configured (that is, if the **path-selection cisco-nondeterministic** statement is not included in the BGP configuration), for paths with the same neighboring AS numbers at the front of the AS path, prefer the path with the lowest MED metric. To always compare MEDs whether or not the peer ASs of the compared routes are the same, include the **path-selection always-compare-med** statement.
- If nondeterministic routing table path selection behavior is configured (that is, the **path-selection cisco-nondeterministic** statement is included in the BGP configuration), prefer the path with the lowest MED metric.

Confederations are not considered when determining neighboring ASs. A missing MED metric is treated as if a MED were present but zero.

NOTE: MED comparison works for single path selection within an AS (when the route does not include an AS path), though this usage Is uncommon.

By default, only the MEDs of routes that have the same peer autonomous systems (ASs) are compared. You can configure routing table path selection options to obtain different behaviors.

8. Prefer strictly internal paths, which include IGP routes and locally generated routes (static, direct, local, and so forth).
9. Prefer strictly external BGP (EBGP) paths over external paths learned through internal BGP (IBGP) sessions.
10. Prefer the path whose next hop is resolved through the IGP route with the lowest metric.

NOTE: A path is considered a BGP equal-cost path (and will be used for forwarding) if a tie-break is performed after the previous step. All paths with the same neighboring AS, learned by a multipath-enabled BGP neighbor, are considered.

BGP multipath does not apply to paths that share the same MED-plus-IGP cost yet differ in IGP cost. Multipath path selection is based on the IGP cost metric, even if two paths have the same MED-plus-IGP cost.

11. If both paths are external, prefer the currently active path to minimize route-flapping. This rule is not used if any one of the following conditions is true:
    - **path-selection external-router-id** is configured.
    - Both peers have the same router ID.

- Either peer is a confederation peer.
- Neither path is the current active path.

12. Prefer a primary route over a secondary route. A primary route is one that belongs to the routing table. A secondary route is one that is added to the routing table through an export policy.
13. Prefer the path from the peer with the lowest router ID. For any path with an originator ID attribute, substitute the originator ID for the router ID during router ID comparison.
14. Prefer the path with the shortest cluster list length. The length is 0 for no list.
15. Prefer the path from the peer with the lowest peer IP address.

## Routing Table Path Selection

The third step of the algorithm, by default, evaluates the length of the AS path and determines the active path. You can configure an option that enables Junos OS to skip this third step of the algorithm by including the **as-path-ignore** option.

NOTE: **The as-path-ignore option is not supported for routing instances.**

To configure routing table path selection behavior, include the **path-selection** statement:

```
path-selection {
  (always-compare-med | cisco-non-deterministic | external-router-id);
  as-path-ignore;
  med-plus-igp {
    igp-multiplier number;
    med-multiplier number;
  }
}
```

For a list of hierarchy levels at which you can include this statement, see the statement summary section for this statement.

Routing table path selection can be configured in one of the following ways:

- Emulate the Cisco IOS default behavior (**cisco-non-deterministic**). This mode evaluates routes in the order that they are received and does not group them according to their neighboring AS. With **cisco-non-deterministic** mode, the active path is always first. All inactive, but eligible, paths follow the active path and are maintained in the order in which they were received, with the most recent path first. Ineligible paths remain at the end of the list.

  As an example, suppose you have three path advertisements for the 192.168.1.0 /24 route:

  - Path 1—learned through EBGP; AS Path of 65010; MED of 200
  - Path 2—learned through IBGP; AS Path of 65020; MED of 150; IGP cost of 5
  - Path 3—learned through IBGP; AS Path of 65010; MED of 100; IGP cost of 10

These advertisements are received in quick succession, within a second, in the order listed. Path 3 is received most recently, so the routing device compares it against path 2, the next most recent advertisement. The cost to the IBGP peer is better for path 2, so the routing device eliminates path 3 from contention. When comparing paths 1 and 2, the routing device prefers path 1 because it is received from an EBGP peer. This allows the routing device to install path 1 as the active path for the route.

NOTE: **We do not recommend using this configuration option in your network. It is provided solely for interoperability to allow all routing devices in the network to make consistent route selections.**

- Always comparing MEDs whether or not the peer ASs of the compared routes are the same (**always-compare-med**).
- Override the rule that If both paths are external, the currently active path is preferred (**external-router-id**). Continue with the next step (Step 12) in the path-selection process.
- Adding the IGP cost to the next-hop destination to the MED value before comparing MED values for path selection (**med-plus-igp**).

  BGP multipath does not apply to paths that share the same MED-plus-IGP cost, yet differ in IGP cost. Multipath path selection is based on the IGP cost metric, even if two paths have the same MED-plus-IGP cost.

### Effects of Advertising Multiple Paths to a Destination

BGP advertises only the active path, unless you configure BGP to advertise multiple paths to a destination.

Suppose a routing device has in its routing table four paths to a destination and is configured to advertise up to three paths (**add-path send path-count 3**). The three paths are chosen based on path selection criteria. That is, the three best paths are chosen in path-selection order. The best path is the active path. This path is removed from consideration and a new best path is chosen. This process is repeated until the specified number of paths is reached.

**Related Documentation**

- Example: Ignoring the AS Path Attribute When Selecting the Best Path
- Examples: Configuring BGP MED
- Example: Advertising Multiple BGP Paths to a Destination

## Understanding Route Preference Values

The Junos OS routing protocol process assigns a default preference value (also known as an *administrative distance*) to each route that the routing table receives. The default value depends on the source of the route. The preference value is a value from 0 through 4,294,967,295 ($2^{32} - 1$), with a lower value indicating a more preferred route. Table 3 on page 27 lists the default preference values.

### Table 3: Default Route Preference Values

| How Route Is Learned | Default Preference | Statement to Modify Default Preference |
|---|---|---|
| Directly connected network | 0 | – |
| System routes | 4 | – |
| Static and Static LSPs | 5 | static |
| RSVP-signaled LSPs | 7 | RSVP **preference** as described in the Junos OS MPLS Applications Configuration Guide |
| LDP-signaled LSPs | 9 | LDP **preference**, as described in the Junos OS MPLS Applications Configuration Guide |
| OSPF internal route | 10 | OSPF preference |
| IS-IS Level 1 internal route | 15 | IS-IS preference |
| IS-IS Level 2 internal route | 18 | IS-IS preference |
| Redirects | 30 | – |
| Kernel | 40 | – |
| SNMP | 50 | – |
| Router discovery | 55 | – |
| RIP | 100 | RIP preference |
| RIPng | 100 | RIPng preference |
| PIM | 105 | Multicast Protocols Configuration Guide |
| DVMRP | 110 | Multicast Protocols Configuration Guide |
| Aggregate | 130 | aggregate |
| OSPF AS external routes | 150 | OSPF external-preference |
| IS-IS Level 1 external route | 160 | IS-IS external-preference |
| IS-IS Level 2 external route | 165 | IS-IS external-preference |
| BGP | 170 | BGP preference, export, import |
| MSDP | 175 | Multicast Protocols Configuration Guide |

In general, the narrower the scope of the statement, the higher precedence its preference value is given, but the smaller the set of routes it affects. To modify the default preference value for routes learned by routing protocols, you generally apply routing policy when configuring the individual routing protocols. You also can modify some preferences with other configuration statements, which are indicated in the table.

**Related Documentation**

- Routing Policy Configuration Guide

## IPv6 Standards

IPv6 is defined in the following documents:

- RFC 1981, *Path MTU Discovery for IP version 6*
- RFC 2373, *IP Version 6 Addressing Architecture*
- RFC 2460, *Internet Protocol, Version 6 (IPv6)*
- RFC 2461, *Neighbor Discovery for IP Version 6*
- RFC 2462, *IPv6 Stateless Address Auto configuration*
- RFC 2463, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6*
- RFC 2464, *Transmission of IPv6 Packets over Ethernet Networks*
- RFC 2472, *IP Version 6 over PPP*
- RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*
- RFC 2675, *IPv6 Jumbo grams*
- RFC 2767, *Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)*
- RFC 2878, *PPP Bridging Control Protocol*
- RFC 2893, *Transition Mechanisms for IPv6 Hosts and Routers*
- Internet draft draft-ietf-dhc-dhcpv6-16.txt, *Dynamic Host Configuration Protocol for IPv6* (expires May 2001)
- Internet draft draft-kato-bgp-ipv6-link-local-00.txt, *BGP4+ Peering Using IPv6 Link-local Address* (expires April 2002)
- Internet draft draft-ietf-idr-flow-spec-00.txt, *Dissemination of Flow Specification Rules*

To access Internet Requests for Comments (RFCs) and drafts, see **http://www.ietf.org**.

CHAPTER 3

# Complete Routing and Routing Protocol Configuration Statements



## [edit logical-systems] Hierarchy Level

The following lists the statements that can be included at the **[edit logical-systems]** hierarchy level and are also documented in this manual.

```
logical-systems {
    logical-system-name {
        protocols {
            bgp {
                bgp-configuration;
            }
            isis {
                isis-configuration;
            }
            ospf {
                ospf-configuration;
            }
            ospf3 {
                ospf3-configuration;
            }
            rip {
                rip-configuration;
            }
            ripng {
                ripng-configuration;
            }
            router-advertisement {
                router-advertisement-configuration;
            }
            router-discovery {
                router-discovery-configuration;
            }
```

```
    }
    routing-instances {
      routing-instance-name {
        routing-instance-configuration;
      }
    }
    routing-options {
      routing-option-configuration;
    }
  }
}
```

## [edit protocols] Hierarchy Level

The following statements can also be included at the **[edit logical-systems *logical-system-name*]** hierarchy level.

**BGP Global**

```
protocols {
  bgp {
    accept-remote-nexthop;
    advertise-external <conditional>;
    advertise-inactive;
    (advertise-peer-as | noadvertise-peer-as);
    authentication-algorithm algorithm;
    authentication-key key;
    authentication-key-chain key-chain;
    bfd-liveness-detection{
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
      }
      holddown-interval milliseconds;
      minimum-interval milliseconds;
      minimum-receive-interval milliseconds;
      no-adaptation;
      session-mode (automatic | multihop | single-hop);
      transmit-interval {
        threshold milliseconds;
        minimum-interval milliseconds;
      }
      multiplier number;
      version (1 | automatic);
    }
    cluster cluster-identifier;
    damping;
    description text-description;
    disable;
    export [ policy-names ];
    family family-name{
    ... family-configuration ...
    graceful-restart {
```

```
    disable;
    restart-time seconds;
    stale-routes-time seconds;
  }
  group group-name {
  ... group-configuration ...
  }
  hold-time seconds;
  idle-after-switch-over (seconds | forever);
  import [ policy-names ];
  include-mp-next-hop;
  ipsec-sa ipsec-sa;
  keep (all | none);
  local-address address;
  local-as autonomous-system <private>;
  local-interface interface-name;
  local-preference local-preference;
  log-updown;
  metric-out (metric | igp (delay-med-update | <metric-offset>) | minimum-igp
    <metric-offset>);
  mtu-discovery;
  multihop {
    no-nexthop-change;
    ttl ttl-value;
  }
  no-aggregator-id;
  no-client-reflect;
  outbound-route-filter{
    bgp-orf-cisco-mode;
    prefix-based {
      accept {
        (inet | inet6);
      }
    }
  }
  out-delay seconds;
  passive;
  path-selection {
    (cisco-non-deterministic | always-compare-med | external-router-id);
    med-plus-igp {
      igp-multiplier number;
      med-multiplier number;
    }
  }
  peer-as autonomous-system;
  preference preference;
  remove-private;
  tcp-mss segment-size;
  traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag <flag-modifier> <disable>;
  }
  vpn-apply-export;
}
```

**BGP Family**

```
family {
  (inet | inet6 | inet-vpn | inet6-vpn | iso-vpn) {
    (any | flow | labeled-unicast | multicast | unicast) {
      accepted-prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      add-path {
        send {
          path-count number;
          prefix-policy [ policy-names ];
          }
        receive;
      }
      <loops number>;
      prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      rib-group group-name;
    }
    flow{
      no-validate policy-name;
    }
    labeled-unicast {
      accepted-prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      aggregate-label {
        community community-name:
      }
      explicit-null {
        connected-only;
      }
      prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      resolve-vpn;
      rib (inet.3 | inet6.3);
      rib-group group-name;
    }
  }
  route-target {
    accepted-prefix-limit {
      maximum number;
      teardown <percentage> <idle-timeout (forever | minutes)>;
    }
    advertise-default;
    external-paths number;
    prefix-limit {
      maximum number;
      teardown <percentage> <idle-timeout (forever | minutes)>;
    }
  }
```

```
    (inet-mdt | inet-mvpn | inet6-mvpn | l2-vpn) {
      signaling {
        accepted-prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        <loops number>;
        prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
          }
        rib-group group-name
      }
    }
  }
```

**BGP Group**

```
  group group-name {
    accept-remote-nexthop;
    advertise-external <conditional>;
    advertise-inactive;
    advertise-peer-as;
    allow ([ network/mask-length ] | all);
    as-override;
    authentication-algorithm algorithm;
    authentication-key key;
    authentication-key-chain key-chain;
    bfd-liveness-detection {
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
      }
      holddown-interval milliseconds;
      minimum-interval milliseconds;
      minimum-receive-interval milliseconds;
      multiplier number;
      no-adaptation;
      session-mode (automatic | multihop | single-hop);transmit-interval {
        threshold milliseconds;
        minimum-interval milliseconds;
      }
      version (1 | automatic);
    }
    cluster cluster-identifier;
    damping;
    description text-description;
    export [ policy-names ];
    family {
      (inet | inet6 | inet-vpn | inet6-vpn | iso-vpn) {
        (any | flow | labeled-unicast | multicast | unicast) {
          accepted-prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
```

```
        }
        add-path {
          send {
            path-count number;
            prefix-policy [ policy-names ];
            }
          receive;
        }
        <loops number>;
        prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
      }
      flow {
        no-validate policy-name;
      }
      labeled-unicast {
        accepted-prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        aggregate-label {
          community community-name:
        }
        explicit-null {
          connected-only;
        }
        prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        resolve-vpn;
        rib inet.3;
        rib-group group-name;
      }
    }
    route-target {
      accepted-prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      advertise-default;
      external-paths number;
      prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
    }
    (inet-mdt | inet-mvpn | inet6-mvpn | l2-vpn) {
      signaling {
        accepted-prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
```

```
            <loops number>;
            prefix-limit {
              maximum number;
              teardown <percentage> <idle-timeout (forever | minutes)>;
              }
            rib-groupgroup-name
          }
        }
      }
      graceful-restart {
        disable;
        restart-time seconds;
        stale-routes-time seconds;
      }
      hold-time seconds;
      idle-after-switch-over (seconds | forever);
      import [ policy-names ];
      include-mp-next-hop;
      ipsec-sa ipsec-sa;
      keep (all | none);
      local-address address;
      local-as autonomous-system <private>;
      local-interface;
      local-preference local-preference;
      log-updown;
      metric-out (metric | minimum-igp <offset> | igp <offset>);
      mtu-discovery;
      multihop <ttl-value>;
      multipath {
        multiple-as;
      }
      no-advertise-peer-as;
      no-aggregator-id;
      no-client-reflect;
      outbound-route-filter {
        bgp-orf-cisco-mode;
        prefix-based {
          accept {
            (inet | inet6);
          }
        }
      }
      out-delay seconds;
      passive;
      peer-as autonomous-system;
      preference preference;
      remove-private;
      tcp-mss segment-size;
      traceoptions {
        file filename <files number> <size size> <world-readable | no-world-readable>;
        flag flag <flag-modifier> <disable>;
      }
      type type;
    vpn-apply-export;
```

**BGP Neighbor**

```
neighbor address {
  accept-remote-nexthop;
  advertise-external <conditional>;
  advertise-inactive;
  advertise-peer-as;
  as-override;
  authentication-algorithm algorithm;
  authentication-key key;
  authentication-key-chain key-chain;
  bfd-liveness-detection {
    authentication {
      algorithm algorithm-name;
      key-chain key-chain-name;
      loose-check;
    }
    detection-time {
      threshold milliseconds;
    }
    minimum-interval milliseconds;
    minimum-receive-interval milliseconds;
    multiplier number;
    no-adaptation;
    session-mode (automatic | multihop | single-hop);
    transmit-interval {
      threshold milliseconds;
      minimum-interval milliseconds;
    }
    version (1 | automatic);
  }
  cluster cluster-identifier;
  damping;
  description text-description;
  export [ policy-names ];
  family{
    (inet | inet6 | inet-vpn | inet6-vpn | iso-vpn) {
      (any | flow | labeled-unicast | multicast | unicast) {
        accepted-prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        add-path {
          send {
            path-count number;
            prefix-policy [ policy-names ];
            }
          receive;
        }
        <loops number>;
        prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
      }
      flow {
        no-validate policy-name;
```

```
    }
    labeled-unicast {
      accepted-prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      aggregate-label {
        community community-name:
      }
      explicit-null {
        connected-only;
      }
      prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      resolve-vpn;
      rib inet.3;
      rib-group group-name;
    }
  }
  route-target {
    accepted-prefix-limit {
      maximum number;
      teardown <percentage> <idle-timeout (forever | minutes)>;
    }
    advertise-default;
    external-paths number;
    prefix-limit {
      maximum number;
      teardown <percentage> <idle-timeout (forever | minutes)>;
    }
  }
  (inet-mdt | inet-mvpn | inet6-mvpn | l2-vpn) {
    signaling {
      accepted-prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
      }
      <loops number>;
      prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | minutes)>;
        }
      rib-group group-name
    }
  }
}
graceful-restart {
  disable;
  restart-time seconds;
  stale-routes-time seconds;
}
hold-time seconds;
idle-after-switch-over (seconds | forever);
import [ policy-names ];
```

```
    include-mp-next-hop;
    ipsec-sa ipsec-sa;
    keep (all | none);
    local-address address;
    local-as autonomous-system <private>;
    local-interface interface-name;
    local-preference local-preference;
    log-updown;
    metric-out (metric | minimum-igp <offset> | igp <offset>);
    mtu-discovery;
    multihop <ttl-value>;
    multipath {
      multiple-as;
    }
    no-advertise-peer-as;
    no-aggregator-id;
    no-client-reflect;
    out-delay seconds;
    outbound-route-filter {
      bgp-orf-cisco-mode;
      prefix-based {
        accept {
          (inet | inet6);
        }
      }
    }
    passive;
    peer-as autonomous-system;
    preference preference;
    remove-private;
    tcp-mss segment-size;
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag <flag-modifier> <disable>;
    }
  vpn-apply-export;
```

IS-IS

```
isis {
  clns-routing;
  disable;
  export [ policy-names ];
  graceful-restart {
    disable;
    helper-disable;
    restart-duration seconds;
  }
  ignore-attached-bit;
  interface (all | interface-name) {
    bfd-liveness-detection {
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
```

```
    }
    minimum-interval milliseconds;
    minimum-receive-interval milliseconds;
    transmit-interval {
      threshold milliseconds;
      minimum-interval milliseconds;
    }
    multiplier number;
  }
  checksum;
  csnp-interval (seconds | disable);
  disable;
  hello-padding (adaptive | loose | strict);
  ldp-synchronization {
    disable;
    hold-time seconds;
  }
  level level-number {
    disable;
    hello-authentication-key key;
    hello-authentication-key-chain key-chain-name;
    hello-authentication-type authentication;
    hello-interval seconds;
    hold-time seconds;
    ipv4-multicast-metric number;
    ipv6-multicast-metric number;
    ipv6-unicast-metric number;
    metric metric;
    passive;
    priority number;
    te-metric metric;
  }
  link-protection;
  lsp-interval milliseconds;
  mesh-group (value | blocked);
  no-adjacency-holddown;
  no-eligible-backup;
  no-ipv4-multicast;
  no-ipv6-multicast;
  no-ipv6-unicast;
  no-unicast-topology;
  node-link-protection;
  passive;
  point-to-point;
}
label-switched-path namelevel level metric metric;
level level-number {
  authentication-key key;
  authentication-key-chain (Protocols IS-IS) key-chain-name;
  authentication-type authentication;
  external-preference preference;
  ipv6-multicast-metric number;
  no-csnp-authentication;
  no-hello-authentication;
  no-psnp-authentication;
  preference preference;
```

```
      prefix-export-limit number;
      wide-metrics-only;
    }
    loose-authentication-check;
    lsp-lifetime seconds;
    max-areas number;
    no-adjacency-holddown;
    no-authentication-check;
    no-ipv4-routing;
    no-ipv6-routing;
    overload {
      advertise-high-metrics;
      timeout seconds;
    }
    reference-bandwidth reference-bandwidth;
    rib-group {
      inet group--name;
      inet6 group--name;
    }
    spf-options {
      delay milliseconds;
      holddown milliseconds;
      rapid-runs number;
    }
    topologies {
      ipv4-multicast;
      ipv6-multicast;
      ipv6-unicast;
    }
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag <flag-modifier> <disable>;
    }
    traffic-engineering {
      disable;
      family inet {
        shortcuts <ignore-lsp-metrics> {
          multicast-rpf-routes;
        }
      }
      family inet6 {
        shortcuts;
      }
    }
  }
```

OSPF

```
ospf {
  disable;
  export [ policy-names ];
  external-preference preference;
  graceful-restart {
    disable;
    helper-disable;
    notify-duration seconds;
    restart-duration seconds;
  }
```

```
import [ policy-names ];
no-nssa-abr;
overload {
  timeout seconds;
}
preference preference;
reference-bandwidth reference-bandwidth;
rib-group group-name;
sham-link {
  local address;
}
spf-options {
  delay milliseconds;
  holddown milliseconds;
  rapid-runs milliseconds;
}
traffic-engineering {
  accept-unnumbered-interfaces;
  multicast-rpf-routes;
  no-topology;
  shortcuts {
    ignore-lsp-metrics;
    lsp-metric-into-summary;
  }
}
traceoptions {
  file filename <files number> <size size> <world-readable | no-world-readable>;
  flag flag <flag-modifier> <disable>;
}
area area-id {
  area-range network/mask-length <restrict> <exact> <override-metric metric>;
  interface interface-name {
    demand-circuit;
    disable;
    bfd-liveness-detection {
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
      }
      minimum-interval milliseconds;
      minimum-receive-interval milliseconds;
      multiplier number;
      no-adaptation;
      transmit-interval {
        threshold milliseconds;
        minimum-interval milliseconds;
      }
      version (1 | automatic);
    }
    ipsec-sa name;
    }
  no-interface-state-traps;
```

```
        authentication {
          md5 key-id {
            key [ key-values ];
          }
          simple-password key-id;
        }
        dead-interval seconds;
        hello-interval seconds;
        interface-type type;
        ldp-synchronization {
          disable;
          hold-time seconds;
        }
        metric metric;
        neighbor address <eligible>;
        network-summary-export [ policy-names ];
        network-summary-import [ policy-names ];
        passive;
        poll-interval seconds;
        priority number;
        retransmit-interval seconds;
        te-metric metric;
        transit-delay seconds;
      }
      label-switched-path name metric metric;
      nssa {
        area-range network/mask-length <restrict> <exact> <override-metric metric>;
        default-lsa {
          default-metric metric;
          metric-type type;
          type-7;
        }
        (no-summaries | summaries);
      }
      peer-interface interface-name {
        disable;
        dead-interval seconds;
        hello-interval seconds;
        retransmit-interval seconds;
        transit-delay seconds;
      }
      sham-link-remote address {
        ipsec-sa name;
      }
      demand-circuit;
      metric metric;
    }
    stub <default-metric metric> <(no-summaries | summaries)>;
    virtual-link neighbor-id router-id transit-area area-id {
      disable;
      ipsec-sa name;
    }
    authentication {
      md5 key-id;
      simple-password key-id;
    }
```

```
dead-interval seconds;
hello-interval seconds;
retransmit-interval seconds;
transit-delay seconds;
```

OSPFv3

```
ospf3 {
  disable;
  export [ policy-names ];
  external-preference preference;
  import [ policy-names ];
  overload {
    timeout seconds;
  }
  preference preference;
  reference-bandwidth reference-bandwidth;
  rib-group group-name;
  spf-options {
    delay milliseconds;
    holddown milliseconds;
    rapid-runs number;
  }
  traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag <flag-modifier> <disable>;
  }
  area area-id {
    area-range network/mask-length <restrict> <exact> <override-metric metric>;
    interface interface-name {
      disable;
      dead-interval seconds;
      hello-interval seconds;
      ipsec-sa name;
      metric metric;
      neighbor address <eligible>;
      passive;
      priority number;
      retransmit-interval seconds;
      transit-delay seconds;
    }
    inter-area-prefix-export [policy-names ];
    inter-area-prefix-import [ policy-names ];
    nssa {
      area-range network/mask-length <restrict> <exact> <override-metric metric>;
      default-lsa {
        default-metric metric;
        metric-type type;
        type-7;
      }
      (no-summaries | summaries);
    }
    stub <default-metric metric> <(no-summaries | summaries)>;
    virtual-link neighbor-id router-id transit-area area-id {
      disable;
      dead-interval seconds;
      hello-interval seconds;
      ipsec-sa name;
```

```
            retransmit-interval seconds;
            transit-delay seconds;
        }
    }
}
```

RIP

```
rip {
    any-sender;
    authentication-key password;
    authentication-type type;
    (check-zero | no-check-zero);
    graceful-restart {
        disable;
        restart-time seconds;
    }
    holddown seconds;
    import [ policy-names ];
    message-size number;
    metric-in metric;
    receive receive-options;
    rib-group group-name;
    route-timeout seconds;
    send send-options;
    update-interval seconds;
    traceoptions {
        file filename <files number> <size size> <world-readable | no-world-readable>;
        flag flag <flag-modifier> <disable>;
    }
    group group-name {
        bfd-liveness-detection {
            authentication {
                algorithm algorithm-name;
                key-chain key-chain-name;
                loose-check;
            }
            detection-time {
                threshold milliseconds;
            }
            minimum-interval milliseconds;
            minimum-receive-interval milliseconds;
            multiplier number;
            no-adaptation;
            transmit-interval {
                threshold milliseconds;
                minimum-interval milliseconds;
            }
            version (0 | 1 | automatic);
        }
        export [ policy-names ];
        metric-out metric;
        preference preference;
        route-timeout seconds;
        update-interval seconds;
        neighbor neighbor-name {
            authentication-key password;
            authentication-type type;
```

```
            bfd-liveness-detection {
              authentication {
                algorithm algorithm-name;
                key-chain key-chain-name;
                loose-check;
              }
              detection-time {
                threshold milliseconds;
              }
              minimum-interval milliseconds;
              minimum-receive-interval milliseconds;
              transmit-interval {
                threshold milliseconds;
                minimum-interval milliseconds;
              }
              multiplier number;
              version (1 | automatic);
            }
            (check-zero | no-check-zero);
            import [ policy-names ];
            message-size number;
            metric-in metric;
            receive receive-options;
            route-timeout seconds;
            send send-options;
            update-interval seconds;
          }
        }
      }
```

**RIPng**

```
ripng {
  graceful-restart {
    disable;
    restart-time seconds;
  }
  holddown seconds;
  import [ policy-names ];
  metric-in metric;
  receive <none>;
  route-timeout seconds;
  send <none>;
  update-interval seconds;
  traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag <flag-modifier> <disable>;
  }
  group group-name {
    export [ policy-names ];
    metric-out metric;
    preference number;
    route-timeout seconds;
    update-interval seconds;
    neighbor neighbor-name {
      import [ policy-names ];
      metric-in metric;
      receive <none>;
```

```
        route-timeout seconds;
        send <none>;
        update-interval seconds;
      }
    }
  }
```

**Router Advertisement**

```
router-advertisement {
  interface interface-name {
    current-hop-limit number;
    default-lifetime seconds;
    (link-mtu | no-link-mtu);
    (managed-configuration | no-managed-configuration);
    max-advertisement-interval seconds;
    min-advertisement-interval seconds;
    (other-stateful-configuration | no-other-stateful-configuration);
    prefix prefix {
      (autonomous | no-autonomous);
      (on-link | no-on-link);
      preferred-lifetime seconds;
      valid-lifetime seconds;
    }
    reachable-time milliseconds;
    retransmit-timer milliseconds;
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag <detail> <disable>;
    }
  }
}
```

**Router Discovery**

```
router-discovery {
  disable;
  interface interface-name {
    min-advertisement-interval seconds;
    max-advertisement-interval seconds;
    lifetime seconds;
  }
  address address {
    (advertise | ignore);
    (broadcast | multicast);
    (priority number | ineligible);
  }
}
```

**Secure Neighbor Discovery**

```
neighbor-discovery {
  secure {
    security-level {
      (default | secure-messages-only);
    }
    cryptographic-address {
      key-length number;
      key-pair pathname;
    }
    timestamp {
```

```
        clock-drift number;
        known-peer-window seconds;
        new-peer-window seconds;
      }
      traceoptions {
        file filename <files number> <match regular-expression> <size size> <world-readable |
          no-world-readable>;
        flag flag;
        no-remote-trace;
      }
    }
}
```

## [edit routing-instances] Hierarchy Level

```
routing-instances {
  routing-instance-name {
    bridge-domains bridge-domain-name {
      domain-type bridge;
      <vlan-id (all | none | number)>;
      <vlan-tags outer number inner number>;
      <routing-interface routing-interface-name>;
      interface interface-name;
      bridge-options {
        interface-mac-limit limit;
        mac-statistics;
        mac-table-size limit;
        no-mac-learning;
        static-mac mac-address;
      }
    }
    description text;
    forwarding-options;
    interface interface-name;
    instance-type (forwarding | layer2–control | l2vpn | no-forwarding | virtual-router |
      virtual-switch | vpls | vrf);
    no-vrf-advertise;
    no-vrf-propagate-ttl;
    route-distinguisher (as-number:number | ip-address:number);
    vrf-import [ policy-names ];
    vrf-export [ policy-names ];
    vrf-propagate-ttl;
    vrf-table-label;
    vrf-target {
      export community-name;
      import community-name;
    }
    protocols {
      bgp {
        bgp-configuration;
      }
      igmp-snooping {
        igmp-snooping-configuration;
      }
      isis {
```

```
    isis-configuration;
  }
  l2vpn {
    l2vpn-configuration;
  }
  ldp {
    ldp-configuration;
  }
  msdp {
    msdp-configuration;
  }
  ospf {
    domain-id domain-id;
    domain-vpn-tag number;
    route-type-community (vendor | iana);
    ospf-configuration;
  }
  ospf 3 {
    domain-id domain-id;
    domain-vpn-tag number;
    route-type-community (vendor | iana);
    ospf3-configuration;
  }
  pim {
    pim-configuration;
  }
  rip {
    rip-configuration;
  }
  vpls {
    vpls-configuration;
  }
}
routing-options {
  aggregate {
    defaults {
      ... aggregate-options ...
    }
    route destination-prefix {
      policy policy-name;
      ... aggregate-options ...
    }
  }
  auto-export {
    (disable | enable);
    family {
      inet {
        flow {
          (disable | enable);
          rib-group rib-group;
        }
        multicast {
          (disable | enable);
          rib-group rib-group;
        }
        unicast {
```

```
            (disable | enable);
            rib-group rib-group;
          }
        }
      }
      traceoptions {
        file filename <files number> <size size> <world-readable | no-world-readable>;
        flag flag <flag-modifier> <disable>;
      }
    }
    autonomous-system autonomous-system <loops number> {
      independent-domain <no-attrset>;
    }
    confederation confederation-autonomous-systems
    members autonomous-system;
    dynamic-tunnels tunnel-name {
      destination-prefix prefix;
      source-address address;
    }
    fate-sharing {
      group group-name;
      cost value;
      from address {
        to address;
      }
      flow {
      route name {
        match {
          match-conditions;
        }
        then {
          actions;
        }
      }
      validation {
        traceoptions {
          file filename <files number> <size size> <world-readable | no-world-readable>;
          flag flag <flag-modifier> <disable>;
        }
      }
      generate {
        defaults {
          generate-options;
        }
        route destination-prefix {
          policy policy-name;
          generate-options;
        }
      }
      instance-export [ policy-names ];
      instance-import [ policy-names ];
      interface-routes {
        family (inet | inet6) {
          export {
            lan;
            point-to-point;
```

```
    }
  }
  rib-group group-name;
}
martians {
  destination-prefix match-type <allow>;
}
maximum-paths path-limit <log-only | threshold value log-interval seconds>;
maximum-prefixes prefix-limit <log-only | threshold value log-interval seconds>;
multicast {
  forwarding-cache {
    threshold (suppress | reuse) value value;
  }
  interface interface-name {
    enable;
  }
  scope scope-name {
    interface interface-name;
    prefix destination-prefix;
  }
  scope-policy policy-name;
  ssm-groups {
    addresses;
  }
}
options {
  syslog (level level | upto level);
}
resolution {
  rib routing-table-name {
    import [ policy-names ];
    resolution-ribs [ routing-table-names ];
  }
}
rib routing-table-name {
  aggregate {
    defaults {
      ... aggregate-options ...
    }
    route destination-prefix {
      policy policy-name;
      ... aggregate-options ...
    }
  }
  filter {
    input filter-name;
  }
  generate {
    defaults {
      generate-options;
    }
    route destination-prefix {
      policy policy-name;
      generate-options;
    }
  }
```

```
    martians {
      destination-prefix match-type <allow>;
    }
    static {
      defaults {
        static-options;
      }
      passive group-name;
      route destination-prefix {
        lsp-next-hop {
          metric metric;
          preference preference;
        }
        next-hop;
        p2mp-lsp-next-hop {
          metric metric;
          preference preference;
        }
        qualified-next-hop address {
          interface interface-name;
          metric metric;
          preference preference;
        }
        static-options;
      }
    }
}
passive {
  group-name {
    import-policy [ policy-names ];
    import-rib [ group-names ];
    export-rib group-name;
  }
}
route-distinguisher-id address;
route-record;
router-id address;
static {
  defaults {
    static-options;
  }
  rib-group group-name;
  route destination-prefix {
    bfd-liveness-detection {
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
      }
      local-address ip-address;
      minimum-interval milliseconds;
      minimum-receive-interval milliseconds;
      minimum-receive-ttl milliseconds;
```

```
            transmit-interval {
              threshold milliseconds;
              minimum-interval milliseconds;
            }
            multiplier number;
            version (1 | automatic);
          }
          lsp-next-hop {
            metric metric;
            preference preference;
          }
          next-hop;
          qualified-next-hop address {
            interface interface-name;
            metric metric;
            preference preference;
          }
          static-options;
        }
      }
      traceoptions {
        file filename <files number> <size size> <world-readable | no-world-readable>;
        flag flag <flag-modifier> <disable>;
      }
    }
  }
}
```

The following statements can also be included at the **[edit logical-systems *logical-system-name*]** hierarchy level.

NOTE: The **virtual-switch** instance type is not supported at the **[edit logical-systems *logical-system-name*]** hierarchy level. For more detailed information about configuring a virtual switch on MX Series routers, see the Junos OS Layer 2 Configuration Guide.

## [edit routing-options] Hierarchy Level

The following statements can also be included at the **[edit logical-systems *logical-system-name*]** hierarchy level.

```
routing-options {
  aggregate {
    defaults {
      ... aggregate-options ...
    }
    route destination-prefix {
      policy policy-name;
      ... aggregate-options ...
    }
  }
```

```
auto-export {
  (disable | enable);
  family {
    inet {
      flow {
        (disable | enable);
        rib-group rib-group;
      }
      multicast {
        (disable | enable);
        rib-group rib-group;
      }
      unicast {
        (disable | enable);
        rib-group rib-group;
      }
    }
  }
  traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag <flag-modifier> <disable>;
  }
}
autonomous-system autonomous-system <loops number>;
no-bfd-triggered-local-repair;
confederation confederation-autonomous-system members autonomous-system;
dynamic-tunnels tunnel-name {
  destination-prefix prefix;
  source-address address;
}
fate-sharing {
  group group-name;
  cost value;
  from address {
    to address;
  }
}
flow {
  route name {
    match {
      match-conditions;
    }
    then {
      actions;
    }
  }
  validation {
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag <flag-modifier> <disable>;
    }
  }
  forwarding-table {
    export [ policy-names ];
    (indirect-next-hop | no-indirect-next-hop);
    unicast-reverse-path (active-paths | feasible-paths);
```

```
}
generate {
  defaults {
    generate-options;
  }
  route destination-prefix {
    policy policy-name;
    generate-options;
  }
}
graceful-restart {
  disable;
  restart-duration seconds;
}
instance-export [ policy-names ];
instance-import [ policy-names ];
interface-routes {
  family (inet | inet6) {
    export {
      lan;
      point-to-point;
    }
  }
  rib-group group-name;
}
martians {
  destination-prefix match-type <allow>;
}
maximum-paths path-limit <log-only | threshold value log-interval seconds>;
maximum-prefixes prefix-limit <log-only | threshold value log-interval seconds>;
multicast {
  forwarding-cache {
    threshold (suppress | reuse) value value;
  }
  interface interface-name {
    enable;
  }
  scope scope-name {
    interface interface-name;
    prefix destination-prefix;
  }
  scope-policy policy-name;
  ssm-groups {
    address;
  }
}
options {
  syslog (level level | upto level);
}
ppm {
  delegate-processing;
}
resolution {
  rib routing-table-name {
    import [ policy-names ];
    resolution-ribs [ routing-table-names ];
```

```
    }
  }
  rib routing-table-name {
    aggregate {
      defaults {
        ... aggregate-options ...
      }
      rib-group group-name;
      route destination-prefix {
        policy policy-name;
        ... aggregate-options ...
      }
    }
    filter {
      input filter-name;
    }
    generate {
      defaults {
        generate-options;
      }
      route destination-prefix {
        policy policy-name;
        generate-options;
      }
    }
    martians {
      destination-prefix match-type <allow>;
    }
    static {
      defaults {
        static-options;
      }
      rib-group group-name;
      route destination-prefix {
        lsp-next-hop {
          metric metric;
          preference preference;
        }
        next-hop;
        qualified-next-hop address {
          interface interface-name;
          metric metric;
          preference preference;
        }
        static-options;
      }
    }
  }
  rib-groups {
    group-name {
      import-policy [ policy-names ];
      import-rib [ group-names ];
      export-rib group-name;
    }
  }
  route-distinguisher-id address;
```

```
route-record;
router-id address;
static {
  defaults {
    static-options;
  }
  rib-group group-name;
  route destination-prefix {
    bfd-liveness-detection {
      authentication {
        algorithm algorithm-name;
        key-chain key-chain-name;
        loose-check;
      }
      detection-time {
        threshold milliseconds;
      }
      holddown-interval milliseconds;
      local-address ip-address;
      minimum-interval milliseconds;
      minimum-receive-interval milliseconds;
      minimum-receive-ttl number;
      multiplier number;
      neighbor address;
      no-adaptation;
      transmit-interval {
        threshold milliseconds;
        minimum-interval milliseconds;
      }
      version (1 | automatic);
    }
    lsp-next-hop {
      metric metric;
      preference preference;
    }
    next-hop;
    p2mp-lsp-next-hop {
      metric metric;
      preference preference;
    }
    qualified-next-hop (address | interface-name) {
      interface interface-name;
      metric metric;
      preference preference;
    }
    source-routing {
      (ip | ipv6);
    }
    static-options;
  }
}
topologies {
  (inet | inet6) {
    topology name;
  }
}
```

```
    traceoptions {
      file filename <files number> <size size> <world-readable | no-world-readable>;
      flag flag <flag-modifier> <disable>;
    }
  }
}
```

PART 3

# Troubleshooting

CHAPTER 4

# Routing Protocol Process Memory FAQs



## Routing Protocol Process Memory FAQs Overview

Junos OS is based on the FreeBSD Unix operating system. The open source software is modified and hardened to operate in the device's specialized environment. For example, some executables have been deleted, while other utilities were de-emphasized. Additionally, certain software processes were added to enhance the routing functionality. The result of this transformation is the kernel, the heart of the Junos OS software.

The kernel is responsible for operating multiple processes that perform the actual functions of the device. Each process operates in its own protected memory space, while the communication among all the processes is still controlled by the kernel. This separation provides isolation between the processes, and resiliency in the event of a process failure. This is important in a core routing platform because a single process failure does not cause the entire device to cease functioning.

Some of the common software processes include the routing protocol process (rpd) that controls the device's protocols, the device control process (dcd) that controls the device's interfaces, the management process (mgd) that controls user access to the device, the chassis process (chassisd) that controls the device's properties itself, and the Packet Forwarding Engine process (pfed) that controls the communication between the device's Packet Forwarding Engine and the Routing Engine. The kernel also generates specialized processes as needed for additional functionality, such as SNMP, the Virtual Router Redundancy Protocol (VRRP), and Class of Service (CoS).

The routing protocol process is a software process within the Routing Engine software, which controls the routing protocols that run on the device. Its functionality includes all protocol messages, routing table updates, and implementation of routing policies.

The routing protocol process starts all configured routing protocols and handles all routing messages. It maintains one or more routing tables, which consolidate the routing information learned from all routing protocols. From this routing information, the routing protocol process determines the active routes to network destinations and installs these routes into the Routing Engine's forwarding table. Finally, it implements routing policy, which allows you to control the routing information that is transferred between the routing

protocols and the routing table. Using routing policy, you can filter and limit the transfer of information as well as set properties associated with specific routes.

**Related Documentation**

- Routing Protocol Process Memory FAQs on page 62

## Routing Protocol Process Memory FAQs

The following sections present the most frequently asked questions and answers related to the routing protocol process memory utilization, operation, interpretation of related command outputs, and troubleshooting the software process.

### Frequently Asked Questions: Routing Protocol Process Memory

This section presents frequently asked questions and answers related to the memory usage of the routing protocol process.

**Why does the routing protocol process use excessive memory?**

The routing protocol process uses hundreds of megabytes of RAM in the Routing Engine to store information needed for the operation of routing and related protocols, such as BGP, OSPF, IS-IS, RSVP, LDP and MPLS. Such huge consumption of memory is common for the process, as the information it stores includes routes, next hops, interfaces, routing policies, labels, and label-switched paths (LSPs). Because access to the RAM memory is much faster than access to the hard disk, most of the routing protocol process information is stored in the RAM memory instead of using the hard disk space. This ensures that the performance of the routing protocol process is maximized.

**How can I check the amount of memory the routing protocol process is using?**

You can check routing protocol process memory usage by entering the **show system processes** and the **show task memory** Junos OS command-line interface (CLI) operational mode commands.

The **show system processes** command displays information about software processes that are running on the device and that have controlling terminals. The **show task memory** command displays memory utilization for routing protocol tasks on the Routing Engine.

You can check the routing protocol process memory usage by using the **show system processes** command with the **extensive** option. The **show task memory** command displays a report generated by the routing protocol process on its own memory usage. However, this report does not display all the memory used by the process. The value reported by the routing protocol process does not account for the memory used for the **TEXT** and **STACK** segments, or the memory used by the process's internal memory manager. Further, the Resident Set Size value includes shared library pages used by the routing protocol process.

For more information about checking the routing protocol process memory usage, see Check Routing Protocol Process (rpd) Memory Usage .

For more information, see the **show system processes** command and the **show task memory** command.

**I just deleted a large number of routes from the routing protocol process. Why is it still using so much memory?**

The **show system processes extensive** command displays a **RES** value measured in kilobytes. This value represents the amount of program memory resident in the physical memory. This is also known as RSS or Resident Set Size. The **RES** value includes shared library pages used by the process. Any amount of memory freed by the process might still be considered part of the **RES** value. Generally, the kernel delays the migrating of memory out of the **Inact** queue into the **Cache** or **Free** list unless there is a memory shortage. This can lead to large discrepancies between the values reported by the routing protocol process and the kernel, even after the routing protocol process has freed a large amount of memory.

## Frequently Asked Questions: Interpreting Routing Protocol Process-Related Command Outputs

This section presents frequently asked questions and answers about the routing protocol process-related Junos OS command-line interface (CLI) command outputs that are used to display the memory usage of the routing protocol process.

**How do I interpret memory numbers displayed in the show system processes extensive command output?**

The **show system processes extensive** command displays exhaustive system process information about software processes that are running on the device and have controlling terminals. This command is equivalent to the UNIX **top** command. However, the UNIX **top** command shows real-time memory usage, with the memory values constantly changing, while the **show system processes extensive** command provides a snapshot of memory usage in a given moment.

To check overall CPU and memory usage, enter the **show system processes extensive** command. Refer to Table 4 on page 64 for information about the **show system processes extensive** commands output fields.

```
user@host> show system processes extensive
last pid:   544;  load averages:  0.00,  0.00,  0.00    18:30:33
37 processes:  1 running, 36 sleeping

Mem: 25M Active, 3968K Inact, 19M Wired, 184K Cache, 8346K Buf, 202M Free
Swap: 528M Total, 64K Used, 528M Free
  PID USERNAME PRI NICE SIZE    RES STATE    TIME   WCPU    CPU COMMAND
  544 root      30   0   604K   768K RUN      0:00  0.00%  0.00% top
    3 root      28   0     0K    12K psleep   0:00  0.00%  0.00% vmdaemon
    4 root      28   0     0K    12K update   0:03  0.00%  0.00% update
  528 aviva     18   0   660K   948K pause    0:00  0.00%  0.00% tcsh
  204 root      18   0   300K   544K pause    0:00  0.00%  0.00% csh
  131 root      18   0   332K   532K pause    0:00  0.00%  0.00% cron
  186 root      18   0   196K    68K pause    0:00  0.00%  0.00% watchdog
   27 root      10   0   512M 16288K mfsidl   0:00  0.00%  0.00% mount_mfs
    1 root      10   0   620K   344K wait     0:00  0.00%  0.00% init
  304 root       3   0   884K   900K ttyin    0:00  0.00%  0.00% bash
  200 root       3   0   180K   540K ttyin    0:00  0.00%  0.00% getty
  203 root       3   0   180K   540K ttyin    0:00  0.00%  0.00% getty
  202 root       3   0   180K   540K ttyin    0:00  0.00%  0.00% getty
  201 root       3   0   180K   540K ttyin    0:00  0.00%  0.00% getty
  194 root       2   0  2248K  1640K select   0:11  0.00%  0.00% rpd
  205 root       2   0   964K   800K select   0:12  0.00%  0.00% tnp.chassisd
```

```
  189 root      2 -12   352K   740K select   0:03  0.00%  0.00% xntpd
  114 root      2   0   296K   612K select   0:00  0.00%  0.00% amd
  188 root      2   0   780K   600K select   0:00  0.00%  0.00% dcd
  527 root      2   0   176K   580K select   0:00  0.00%  0.00% rlogind
  195 root      2   0   212K   552K select   0:00  0.00%  0.00% inetd
  187 root      2   0   192K   532K select   0:00  0.00%  0.00% tnetd
   83 root      2   0   188K   520K select   0:00  0.00%  0.00% syslogd
  538 root      2   0  1324K   516K select   0:00  0.00%  0.00% mgd
   99 daemon    2   0   176K   492K select   0:00  0.00%  0.00% portmap
  163 root      2   0   572K   420K select   0:00  0.00%  0.00% nsrexecd
  192 root      2   0   560K   400K select   0:10  0.00%  0.00% snmpd
  191 root      2   0  1284K   376K select   0:00  0.00%  0.00% mgd
  537 aviva     2   0   636K   364K select   0:00  0.00%  0.00% cli
  193 root      2   0   312K   204K select   0:07  0.00%  0.00% mib2d
    5 root      2   0     0K    12K pfesel   0:00  0.00%  0.00% if_pfe
    2 root    -18   0     0K    12K psleep   0:00  0.00%  0.00% pagedaemon
    0 root    -18   0     0K     0K sched    0:00  0.00%  0.00% swapper
```

Table 4 on page 64 describes the output fields that represent the memory values for the **show system processes extensive** command. Output fields are listed in the approximate order in which they appear.

### Table 4: show system processes extensive Output Fields

| Field Name | Field Description |
|---|---|
| **Mem** | Information about physical and virtual memory allocation. |
| **Active** | Memory allocated and actively used by the program. |
| **Inact** | Memory allocated but not recently used or memory freed by the programs. Inactive memory remains mapped in the address space of one or more processes and, therefore, counts toward the RSS value of those processes. |
| **Wired** | Memory that is not eligible to be swapped, usually used for in-kernel memory structures and/or memory physically locked by a process. |
| **Cache** | Memory that is not associated with any program and does not need to be swapped before being reused. |
| **Buf** | Size of memory buffer used to hold data recently called from the disk. |
| **Free** | Memory that is not associated with any programs. Memory freed by a process can become **Inactive**, **Cache**, or **Free**, depending on the method used by the process to free the memory. |
| **Swap** | Information about swap memory.<br>• Total—Total memory available to be swapped to disk.<br>• Used—Memory swapped to disk.<br>• Free—Memory available for further swap. |

The rest of the command output displays information about the memory usage of each process. The **SIZE** field indicates the size of the virtual address space, and the **RES** field indicates the amount of the program in physical memory, which is also known as RSS or Resident Set Size. For more information, see the **show system processes** command.

**What is the difference between Active and Inact memory that is displayed by the show system processes extensive command?**

When the system is under memory pressure, the pageout process reuses memory from the free, cache, inact and, if necessary, active pages. When the pageout process runs, it scans memory to see which pages are good candidates to be unmapped and freed up. Thus, the distinction between **Active** and **Inact** memory is only used by the pageout process to determine which pool of pages to free first at the time of a memory shortage.

The pageout process first scans the **Inact** list, and checks whether the pages on this list have been accessed since the time they have been listed here. The pages that have been accessed are moved from the **Inact** list to the **Active** list. On the other hand, pages that have not been accessed become prime candidates to be freed by the pageout process. If the pageout process cannot produce enough free pages from the **Inact** list, pages from the **Active** list get freed up.

Because the pageout process runs only when the system is under memory pressure, the pages on the **Inact** list remain untouched – even if they have not been accessed recently – when the amount of **Free** memory is adequate.

**How do I interpret memory numbers displayed in the show task memory command output?**

The **show task memory** command provides a comprehensive picture of the memory utilization for routing protocol tasks on the Routing Engine. The routing protocol process is the main task that uses Routing Engine memory.

To check routing process memory usage, enter the **show task memory** command. Refer to Table 5 on page 65 for information about the **show task memory** command output fields.

```
user@host> show task memory
Memory                 Size (kB)  %Available  When
  Currently In Use:        29417          3%  now
  Maximum Ever Used:       33882          4%  00/02/11 22:07:03
  Available:              756281        100%  now
```

Table 5 on page 65 describes the output fields for the **show task memory** command. Output fields are listed in the approximate order in which they appear.

### Table 5: show task memory Output Fields

| Field Name | Field Description |
|---|---|
| **Memory Currently In Use** | Memory currently in use. Dynamically allocated memory plus the **DATA** segment memory in kilobytes. |
| **Memory Maximum Ever Used** | Maximum memory ever used. |
| **Memory Available** | Memory currently available. |

The **show task memory** command does not display all the memory used by the routing protocol process. This value does not account for the memory used for the **TEXT** and

**STACK** segments, or the memory used by the routing protocol process's internal memory manager.

**Why is the Currently In Use value less than the RES value?**

The **show task memory** command displays a **Currently In Use** value measured in kilobytes. This value represents the memory currently in use. It is the dynamically allocated memory plus the **DATA** segment memory. The **show system processes extensive** command displays a **RES** value measured in kilobytes. This value represents the amount of program memory resident in the physical memory. This is also known as RSS or Resident Set Size.

The **Currently In Use** value does not account for all of the memory that the routing protocol process uses. This value does not include the memory used for the **TEXT** and the **STACK** segments, and a small percentage of memory used by the routing protocol process's internal memory manager. Further, the **RES** value includes shared library pages used by the routing protocol process.

Any amount of memory freed by the routing protocol process might still be considered part of the **RES** value. Generally, the kernel delays the migrating of memory out of the **Inact** queue into the **Cache** or **Free** list unless there is a memory shortage. This can lead to large discrepancies between the **Currently In Use** value and the **RES** value.

## Frequently Asked Questions: Routing Protocol Process Memory Swapping

This section presents frequently asked questions and answers related to the memory swapping of the routing protocol process from the Routing Engine memory to the hard disk memory.

**How do I monitor swap activity?**

When the system is under memory pressure, the pageout process reuses memory from the free, cache, inact and, if necessary, active pages. You can monitor the swap activity by viewing the syslog message reported by the kernel during periods of high pageout activity.

The syslog message appears as follows:

```
Mar  3 20:08:02 olympic /kernel: High pageout rate!! 277 pages/sec.
```

You can use the **vmstat -s** command to print the statistics for the swapout activity. The displayed statistics appear as follows:

```
0 swap pager pageouts
0 swap pager pages paged out
```

The **swap pager pageouts** is the number of pageout operations to the swap device, and the **swap pager pages paged out** is the number of pages paged out to the swap device.

**Why does the system start swapping when I try to dump core using the request system core-dumps command?**

The **request system core-dumps** command displays a list of system core files created when the device has failed. This command can be useful for diagnostic purposes. Each list item includes the file permissions, number of links, owner, group, size, modification

date, path, and filename. You can use the **core-filename** option and the **core-file-info**, **brief**, and **detail** options to display more information about the specified core-dump files.

You can use the **request system core-dumps** command to perform a non-fatal core-dump without aborting the routing protocol process. To do this, the routing protocol process is forked, generating a second copy, and then aborted. This process can double the memory consumed by the two copies of the routing protocol processes, pushing the system into swap.

**Why does the show system processes extensive command show that memory is swapped to disk although there is plenty of free memory?**

Memory can remain swapped out indefinitely if it is not accessed again. Therefore, the **show system processes extensive** command shows that memory is swapped to disk even though there is plenty of free memory, and such a situation is not unusual.

## Frequently Asked Questions: Troubleshooting the Routing Protocol Process

This section presents frequently asked questions and answers related to a shortage of memory and memory leakage by the routing protocol process.

**What does the RPD_OS_MEMHIGH message mean?**

The **RPD_OS_MEMHIGH** message is written into the system message file if the routing protocol process is running out of memory. This message alerts you that the routing protocol process is using the indicated amount and percentage of Routing Engine memory, which is considered excessive. This message is generated either because the routing protocol process is leaking memory or the use of system resources is excessive, perhaps because routing filters are misconfigured or the configured network topology is very complex.

When the memory utilization for the routing protocol process is using all available Routing Engine DRAM memory (Routing Engines with maximum 2 GB DRAM) or reaches the limit of 2 GB of memory (Routing Engines with 4 GB DRAM), a message of the following form is written every minute in the syslog message file:

```
RPD_OS_MEMHIGH: Using 188830 KB of memory, 100 percent of available
```

This message includes the amount, in kilobytes and/or the percentage, of the available memory in use.

This message should not appear under normal conditions, as any further memory allocations usually require a portion of existing memory to be written to swap. As a recommended solution, increase the amount of RAM in the Routing Engine. For more information, go to http://kb.juniper.net/InfoCenter/index?page=content&id=KB14186 .

**What can I do when there is a memory shortage even after a swap?**

It is not recommended for the system to operate in this state, notwithstanding the existence of swap. The protocols that run in the routing protocol process usually have a real-time requirement that cannot reliably withstand the latency of being swapped to hard disk. If the memory shortage has not resulted from a memory leak, then either a

reduction in the memory usage or an upgrade to a higher memory-capacity Routing Engine is required.

**How do I determine whether there is a memory leak in the routing protocol process?**

Memory leaks are typically the result of a seemingly unbounded growth in the memory usage of a process as reported by the **show system processes extensive** command.

There are two classes of memory leaks that the routing protocol process can experience.

- The first class occurs when the allocated memory that is no longer in use is not freed. This class of leak can usually be fixed by taking several samples of the **show task memory detail** command over a period of time and comparing the deltas.
- The second class occurs when there is a late access to freed memory. If the access is not outside the mapped address space, the kernel backfills the accessed page with real memory. This backfill is done without the knowledge of the routing protocol process's internal memory allocator, which makes this class of leak much more difficult to resolve. If a memory leak of this class is suspected, writing the state of the system to a disk file (creating a core file) is suggested.

A large discrepancy between the **RES** value and the **Currently In Use** value might indicate a memory leak. However, large discrepancies can also occur for legitimate reasons. For example, the memory used for the **TEXT** and **STACK** segments or the memory used by the routing protocol process's internal memory manager might not be displayed. Further, the **RES** value includes shared library pages used by the process.

**What is the task_timer?**

The source of a routing protocol process memory leak can usually be identified by dumping the timers for each task. You can use the **show task *task-name*** command to display routing protocol tasks on the Routing Engine. Tasks can be baseline tasks performed regardless of the device's configuration, and other tasks that depend on the device configuration.

For more information, see the **show task** command.

**Related Documentation**

- Routing Protocol Process Memory FAQs Overview on page 61

PART 4

# Index

# Index