CHAPTER

8

Routing Policy and Firewall Filters

The JUNOS Internet software provides a *policy framework*, which is a collection of JUNOS policies that allows you to control flows of routing information and packets. The policy framework is composed of routing policy, which allows you to control the routing information or change between the routing protocols and the routing tables and between the routing tables and the forwarding table and firewall filter policy, which allows you to control packets transiting the router to a network destination and packets destined for and sent by the router.

NOTE The term *firewall filter policy* is used here to emphasize that a firewall filter is a policy and shares some fundamental similarities with a routing policy. However, when referring to a firewall filter policy in the remainder of this book, the term *firewall filter* is used.

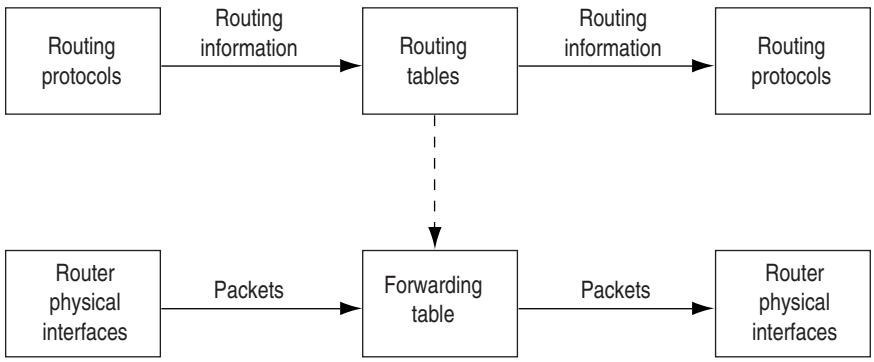
The JUNOS policies affect the following router flows:

- Flow of routing information between the routing protocols and the routing tables and between the routing tables and the forwarding table. The Routing Engine handles this flow. *Routing information* is the information about routes learned by the routing protocols from a router's neighbors. This information is stored in routing tables and is subsequently advertised by the routing protocols to the router's neighbors. Routing policies allow you to control the flow of this information.
- Flow of data packets in and out of the router physical interfaces. The Packet Forwarding Engine handles this flow. *Data packets* are chunks of data that transit the router as they are being forwarded from a source to a destination. When a router receives a data packet on an interface, the router determines where to forward the packet by looking in the forwarding table for the best route to a destination. The router then forwards the data packet toward its destination through the appropriate interface. Firewall filters allow you to control the flow of these data packets.
- Flow of local packets from the router physical interfaces and to the Routing Engine. The Routing Engine handles this flow. *Local packets* are chunks of data that are destined for or sent by the router. Local packets usually contain routing protocol data, data for IP services such as telnet or secure shell (ssh), and data for administrative protocols such as the Internet Control Message Protocol (ICMP). When the Routing Engine receives a local

packet, it forwards the packet to the appropriate daemon or to the kernel, which are both part of the Routing Engine, or to the Packet Forwarding Engine. Firewall filters allow you to control the flow of these local packets.

Figure 8.1 illustrates the flows of routing information and packets through the router. Although the flows are very different from each other, they are also interdependent. Routing policies determine which routes are placed in the forwarding table. The forwarding table in turn has an integral role in determining the appropriate physical interface through which to forward a packet.

Figure 8.1 *Flows of Routing Information and Packets*

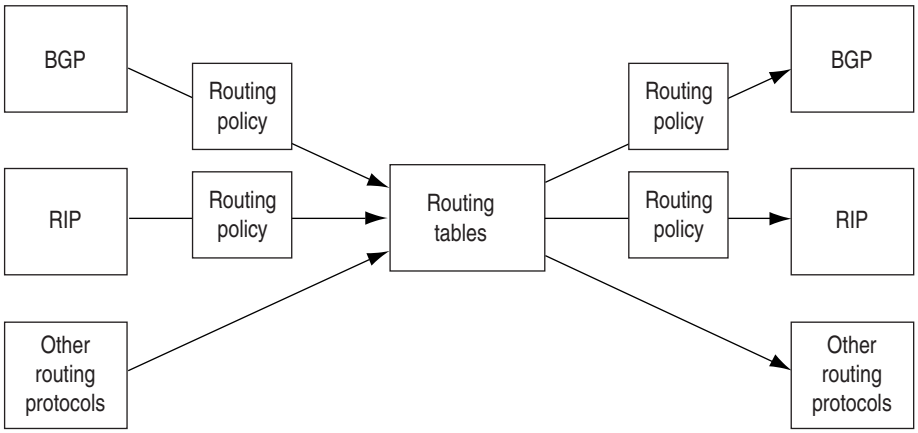


You can configure routing policies to control which routes the routing protocols place in the routing tables and to control which routes the routing protocols advertise from the routing tables (see Figure 8.2). The routing protocols advertise active routes only from the routing tables. (An *active route* is a route that is chosen from all routes in the routing table to reach a destination.)

For more information about the active route selection process, see "How the Active Route Is Determined," on page 376.

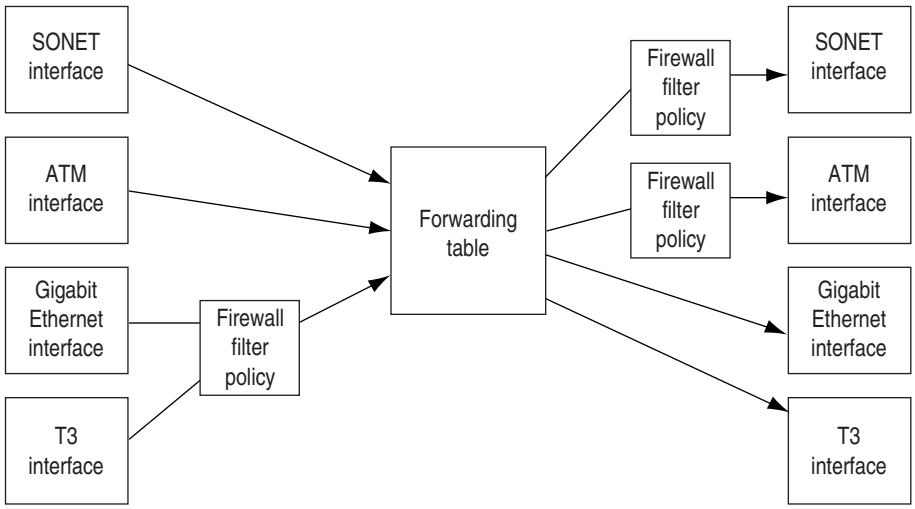
You can also use routing policies to change specific route characteristics, which allow you to control which route is selected as the active route to reach a destination, to effect changes to the default BGP route flap-damping values, to perform per-packet load balancing, and to enable class of service (CoS).

Figure 8.2 *Routing Policies to Control Routing Information Flow*



Firewall filters provide a means of protecting your router from excessive traffic transiting the router to a network destination or destined for the Routing Engine. Firewall filters that control local packets can also protect your router from external aggressions such as denial-of-service (DoS) attacks. You can configure firewall filters to control which data packets are accepted on and transmitted from the physical interfaces and which local packets are transmitted from the physical interfaces to the Routing Engine (see Figure 8.3).

Figure 8.3 *Firewall Filters to Control Packet Flow*

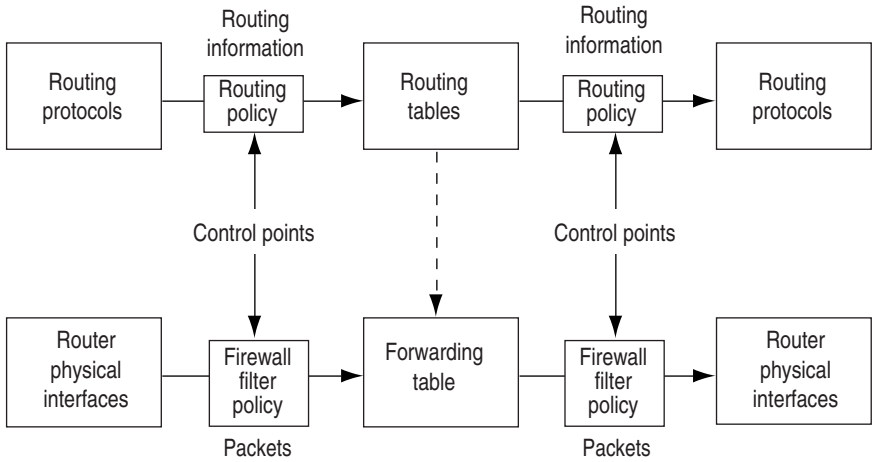


Policy Terminology

A *policy* is a mechanism in the JUNOS policy framework that allows you to configure criteria against which something can be compared and an action that is performed if the criteria are met.

All policies provide two points at which you can control routing information or packets through the router (see Figure 8.4). These *control points* allow you to control routing information before and after it is placed in the routing table, data packets before and after a forwarding table lookup, and local packets before and after they are received by the Routing Engine. (Figure 8.4 appears to depict only one control point, but because of the bidirectional flow of the local packets, two control points actually exist.)

Figure 8.4 *Policy Control Points*



Because there are two control points, you can configure policies that control the routing information or data packets before and after their interaction with their respective tables and local packets before and after their interaction with the Routing Engine. *Import routing policies* control the routing information that is placed in the routing tables, while *export routing policies* control the routing information that is advertised from the routing tables. *Input firewall filters* control packets that are received on a router interface, while *output firewall filters* control packets that are transmitted from a router interface.

All policies consist of the following configurable components:

- Match conditions—Criteria against which a route or packets are compared. You can configure one or more criteria. If all criteria match, one or more actions are applied.
- Actions—What happens if all criteria match. You can configure one or more actions.
- Terms—Named structures in which match conditions and actions are defined. You can define one or more terms.

The policy framework software evaluates each incoming and outgoing route or packet against the match conditions in a term. If the criteria in the match conditions are met, the defined action is taken. In general, the policy framework software compares the route or packet against the match conditions in the first term in the policy, then goes on to the next term, and so on. Therefore, the order in which you arrange terms in a policy is relevant. However, the order of match conditions within a term is not relevant because a route or packet must match all match conditions in a term for an action to be taken.

If an incoming or outgoing route or packet arrives and an explicitly configured policy related to the route or to the interface upon which the packet arrives is not configured, the action specified by the default policy is taken. A *default policy* is a rule or a set of rules that determine whether the route is placed in or advertised from the routing table, or whether the packet is accepted into or transmitted from the router interface. All policies also have default actions in case a policy does not specify a match condition, a match occurs, but a policy does not specify an action, a match does not occur with a term in a policy and subsequent terms in the same policy exist, or a match does not occur by the end of a policy.

All policies share a two-step configuration process:

- Define the policy—Define the policy components, including criteria against which routes or packets are compared and actions that are performed if the criteria are met.
- Apply the policy—Apply the policy to whatever moves the routing information or packets through the router, for example, the routing protocol or router interface.

The JUNOS policy architecture is simple and straightforward. However, the actual implementation of each policy adds layers of complexity to the respective policies as well as power and flexibility to your router's capabilities. Configuring a policy has a major impact on the flow of routing information or packets within and through the router. For example, you can configure a routing policy that does not allow routes associated with a particular customer to be placed in the routing table. As a result of this routing policy, the customer routes are not used to forward data packets to various destinations, and the routes are not advertised by the routing protocol to neighbors. Before configuring a policy, determine what you want to accomplish with it and thoroughly understand how to achieve your goal using the various match conditions and actions. Also, make sure that you understand the default policies and actions for the policy you are configuring.

Comparison of Routing Policies and Firewall Filters

Although routing policies and firewall filters share a common architecture, several differences exist. The fundamental difference between the policies is their purpose, and because of this, the implementation details, and, consequently, the configuration methods for each are very different. Table 8.1 compares the implementation details for routing policies and firewall filters, highlighting the similarities and differences between the two policies.

Table 8.1 *Policy Implementation Details*

Policy Architecture	Routing Policy Implementation	Firewall Filter Implementation
Control points	Control routing information that is placed in the routing table with an import routing policy and advertised from the routing table with an export routing policy.	Control packets that are accepted on a router interface with an input firewall filter and that are forwarded from an interface with an output firewall filter.

Table 8.1 *Policy Implementation Details*

Policy Architecture	Routing Policy Implementation	Firewall Filter Implementation
Configuration tasks: n Define policy n Apply policy	Define a policy that contains terms, match conditions, and actions. Apply one or more export or import policies to a routing protocol. You can also apply a *policy expression*, which uses Boolean logical operators with multiple import or export policies. You can also apply one or more export policies to the forwarding table.	Define a policy that contains terms, match conditions, and actions. Apply one input or output firewall filter to a physical interface or physical interface group to filter data packets received by or forwarded to a physical interface (on routers with an Internet Processor II ASIC only). You can also apply one input or output firewall filter to the router's loopback interface, which is the interface to the Routing Engine (on all routers). Doing this allows you to filter local packets received by or forwarded from the Routing Engine.
Terms	Configure as many terms as desired in a policy. Define a name for each term. Terms are evaluated in the order in which you specify them in a policy. Evaluation of a policy ends after a packet matches the criteria in a term and the defined or default policy action of accept or reject is taken. The route is not evaluated against subsequent terms in the same policy or subsequent policies.	Configure as many terms as desired in a firewall filter. Define a name for each term. Terms are evaluated in the order in which you specify them in a firewall filter. Evaluation of a firewall filter ends after a packet matches the criteria in a term and the defined or default action is taken. The packet is not evaluated against subsequent terms in the firewall filter.
Match conditions	Specify zero or more criteria that a route must match. You can specify criteria based on source, destination, or properties of a route. You can also specify the following match conditions, which require more configuration: n Autonomous system (AS) path expression—A combination of AS numbers and regular expression operators. n Community—A group of destinations that share a common property. n Prefix list—A named list of prefixes. n Route list—A list of destination prefixes. n Subroutine—A routing policy that is called repeatedly from other routing policies.	Specify zero or more criteria that a packet must match. You must match various fields in the packet's header. The fields are grouped into the following categories: n Numeric values, such as port and protocol numbers. n Prefix values, such as IP source and destination prefixes. n Bit-field values, that is, if particular bits in the fields are or are not set, such as IP options, TCP flags, and IP fragmentation fields. You can specify the fields using Boolean logical operators.

Table 8.1 *Policy Implementation Details*

Policy Architecture	Routing Policy Implementation	Firewall Filter Implementation
Actions	Specify zero or one action to take if a route matches all criteria. You can specify the following actions: n Accept—Accept the route into the routing table and propagate it. After this action is taken, the evaluation of subsequent terms and policies ends. n Reject—Do not accept the route into the routing table, and do not propagate it. After this action is taken, the evaluation of subsequent terms and policies ends. In addition to the actions described above, you can also specify zero or more of the following types of actions: n Next term—Evaluate the next term in the routing policy. n Next policy—Evaluate the next routing policy. n Actions that manipulate characteristics associated with a route as the routing protocol places it in the routing table or advertises it from the routing table. n Trace action, which logs route matches.	Specify zero or one action to take if a packet matches all criteria. (Juniper Networks recommends that you always explicitly configure an action.) You can specify the following actions: n Accept—Accept a packet. n Discard—Discard a packet silently, without sending an ICMP message. n Reject—Discard a packet and send an ICMP destination unreachable message. n Routing instance—Specify a routing table to which packets are forwarded. In addition to zero or one of the actions described above, you can also specify zero or more action modifiers. You can specify the following action modifiers: n Count—Add packet to a count total. n Forwarding class—Set the packet forwarding class to a specified value from 0 through 3. n IPSec security association—Used with the source and destination address match conditions, specify an IP Security (IPSec) security association (SA) for the packet. n Log—Store the header information of a packet on the Routing Engine. n Loss priority—Set the packet loss priority (PLP) bit to a specified value, 0 or 1. n Policer—Apply rate-limiting procedures to the traffic. n Sample—Sample the packet traffic. n Syslog—Log an alert for the packet.

Table 8.1 *Policy Implementation Details*

Policy Architecture	Routing Policy Implementation	Firewall Filter Implementation
Default policies and actions	If an incoming or outgoing route arrives and a policy related to the route is not explicitly configured, the action specified by the default policy for the associated routing protocol is taken. The following default actions exist for routing policies: n If a policy does not specify a match condition, all routes evaluated against the policy match. n If a match occurs but the policy does not specify an accept, reject, next term, or next policy action, one of the following occurs: n The next term, if present, is evaluated. n If no other terms are present, the next policy is evaluated. n If no other policies are present, the action specified by the default policy is taken. n If a match does not occur with a term in a policy and subsequent terms in the same policy exist, the next term is evaluated. n If a match does not occur with any terms in a policy and subsequent policies exist, the next policy is evaluated. n If a match does not occur by the end of a policy and no other policies exist, the accept or reject action specified by the default policy is taken.	If an incoming or outgoing packet arrives on an interface and a firewall filter is not configured for the interface, the default policy is taken (the packet is accepted). The following default actions exist for firewall filters: n If a firewall filter does not specify a match condition, all packets are considered to match. n If a match occurs but the firewall filter does not specify an action, the packet is accepted. n If a match occurs, the defined or default action is taken, and the evaluation ends. Subsequent terms in the firewall filter are not evaluated. n If a match does not occur with a term in a firewall filter and subsequent terms in the same filter exist, the next term is evaluated. n If a match does not occur by the end of a firewall filter, the packet is discarded.

Routing Policy Framework

All routing protocols store their routing information in routing tables. From these tables, the routing protocols calculate the best route to each destination and place these routes in a forwarding table. These routes are then used to forward routing protocol traffic toward a destination, and they can be advertised to neighbors using one or more routing protocols. In general, the routing protocols place all their routes in the routing table and advertise a limited set of routes from the routing table. The general rules for handling the routing information between the routing protocols and the routing table are known as the *routing policy framework*.

Instead of referring to the multiple routing tables that the JUNOS software maintains, this discussion assumes the `inet.0` routing table unless explicitly stated otherwise. By default, the JUNOS software stores unicast Internet Protocol Version 4 (IPv4) routes in the `inet.0` routing table.

A *routing policy* is a mechanism in the JUNOS software that allows you to modify the routing policy framework to suit your needs.

The routing policy framework is composed of default rules for each routing protocol that determine which routes the protocol places in the routing table and advertises from the routing table. The default rules for each routing protocol are known as *default routing policies*. You can create routing policies to preempt the default policies, which are always present. A *routing policy* is a mechanism in the JUNOS software that allows you to modify the routing policy framework to suit your needs. You can create and implement your own routing policies to control which routes a routing protocol places in the routing table, control which active routes a routing protocol advertises from the routing table, or manipulate the route characteristics as a routing protocol places it in the routing table or advertises it from the routing table.

You might want to preempt the default routing policies in the routing policy framework by creating your own routing policies in the following circumstances:

- To not have a protocol to import all routes into the routing table. If the routing table does not learn about certain routes, they can never be used to forward packets, and they can never be redistributed into other routing protocols.
- To not have a routing protocol export all the active routes learned by that protocol.

- To have a routing protocol announce active routes learned from another routing protocol, which is sometimes called *route redistribution*.
- To manipulate route characteristics, such as the preference value, AS path, or the community. You can manipulate the route characteristics to control which route is selected as the active route to reach a destination. In general, the active route is also advertised to a router's neighbors.
- To change the default BGP route flap-damping parameters.
- To perform per-packet load balancing.
- To enable class of service (CoS).

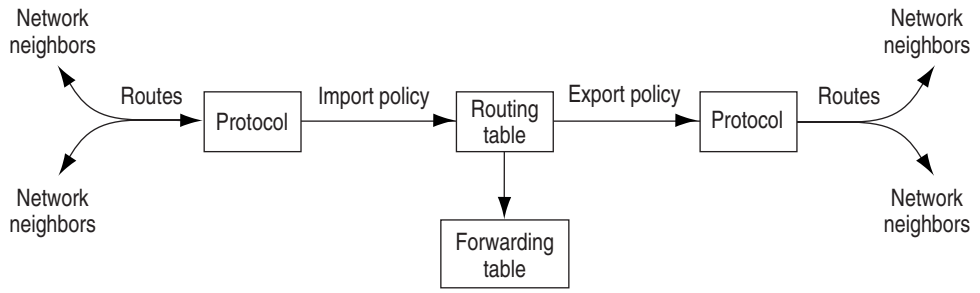
You can manipulate the route characteristics to control which route is selected as the active route to reach a destination. The active route is placed in the forwarding table and used to forward traffic toward the route's destination. In general, the active route is also advertised to a router's neighbors. To create a routing policy, you must define the policy and apply it. You define the policy by specifying the criteria that a route must match and the actions to perform if a match occurs. You then apply the policy to a routing protocol or to the forwarding table.

To better understand the routing policy framework, it is necessary to define two terms that explain how routes move between the routing protocols and the routing table (see Figure 8.5):

- When the Routing Engine places the routes of a routing protocol into the routing table, it is *importing* routes into the routing table.
- When the Routing Engine uses active routes from the routing table to send a protocol advertisement, it is *exporting* routes from the routing table.

The process of moving routes between a routing protocol and the routing table always is described *from the point of view of the routing table*. That is, routes are *imported into* a routing table from a routing protocol, and they are *exported from* a routing table to a routing protocol. Remember this distinction when working with routing policies.

Figure 8.5 *Importing and Exporting Routes*



When evaluating routes for export, the Routing Engine uses only active routes from the routing table. In other words, an export policy does not evaluate all routes; it evaluates only those routes that a routing protocol is allowed to advertise to a neighbor.

Table 8.2 lists the routing protocols from which the routing table can import routes and to which routing protocols the routing table can export routes. This table also lists direct and explicitly configured routes, which for the purposes of this table are considered a pseudoprotocol. An *explicitly configured route* is a route that you have configured. *Direct routes* are not explicitly configured; they are created as a result of IP addresses being configured on an interface. Explicitly configured routes include aggregate, generated, local, and static routes. An *aggregate route* is a route that distills groups of routes with common addresses into one route. A *generated route* is a route used when the routing table has no information about how to reach a particular destination. A *local route* is an IP address assigned to a router interface. A s*tatic route* is a nonchanging route to a destination.

The policy framework software treats direct and explicitly configured routes as if they are learned through routing protocols; therefore, they can be imported into the routing table. Routes cannot be exported from the routing table to the pseudoprotocol because this protocol is not a real routing protocol. However, aggregate, direct, generated, and static routes can be exported from the routing table to routing protocols, whereas local routes cannot.

For information about the default routing policies for each routing protocol, see Table 8.4. For information about the import and export routing policies supported for each routing protocol and the level at which you can apply these policies, see Table 8.6 on page 320.

Table 8.2 *Protocols the Routing Table Can Import from and Export to*

Protocol	Import	Export
Border Gateway Protocol (BGP)	Yes	Yes
Distance Vector Multicast Routing Protocol (DVMRP)	Yes	Yes
Intermediate System to Intermediate System (IS-IS)	Yes	Yes
Label Distribution Protocol (LDP)	Yes	Yes
Multiprotocol Label Switching (MPLS)	Yes	No
Open Shortest Path First (OSPF)	Yes	Yes
Protocol-Independent Multicast (PIM) dense mode	Yes	Yes
PIM sparse mode	Yes	Yes

Pseudo protocols:
n Direct routes
n Explicitly configured routes
  n Aggregate routes
  n Generated routes
  n Local routes
  n Static routes
	Yes	No

Routing Information Protocol (RIP) and Routing Information Protocol Next-Generation (RIPng)	Yes	Yes

Table 8.3 lists the routing tables affected by default and user-defined routing policies and the types of routes that each routing table stores.

Table 8.3 *Routing Tables Affected by Routing Policies*

Routing Table	Type of Routes Stored
`inet.0`	Unicast IPv4 routes
`instance-name``.inet.0`	Unicast IPv4 routes for a particular routing instance
`inet.1`	Multicast IPv4 routes

Table 8.3 *Routing Tables Affected by Routing Policies*

Routing Table	Type of Routes Stored
`inet.2`	Unicast IPv4 routes for multicast reverse-path forwarding (RPF) lookup
`inet.3`	MPLS routes
`mpls.0`	MPLS routes for label-switched path (LSP) next hops
`inet6.0`	Unicast IPv6 routes

Table 8.4 summarizes the default routing policies for each routing protocol that imports and exports routes. The actions in the default routing policies are taken if you have not explicitly configured a routing policy.

Table 8.4 *Default Routing Policies*

Importing or Exporting Protocol	Default Import Policy	Default Export Policy
BGP	Import all BGP IPv4 routes learned from configured neighbors into the `inet.0` routing table. Import all BGP IPv6 routes learned from configured neighbors into the `inet6.0` routing table.	Export active BGP routes.
DVMRP	Import all DVMRP routes into the `inet.1` routing table.	Export active DVMRP routes.
IS-IS	Import all IS-IS routes into the `inet.0` and `inet6.0` routing tables. (You cannot override or change this default policy.)	Export active IS-IS routes. Export direct (interface) routes for the interfaces on which IS-IS is explicitly configured.
LDP	Import all LDP routes into the `inet.3` routing table.	Export active LDP routes.
MPLS	Import all MPLS routes into the `inet.3` routing table.	Export active MPLS routes.
OSPF	Import all OSPF routes into the `inet.0` routing table. (You cannot override or change this default policy.)	Export active OSPF routes. Export direct (interface) routes for the interfaces on which OSPF is explicitly configured.

Table 8.4 *Default Routing Policies*

Importing or Exporting Protocol	Default Import Policy	Default Export Policy
PIM dense mode	Import all PIM dense mode routes into the `inet.1` routing table.	Export active PIM dense mode routes.
PIM sparse mode	Import all PIM sparse mode routes into the `inet.1` routing table.	Export active PIM sparse mode routes.

Pseudoprotocol:
n Direct routes
n Explicitly configured routes:
  n Aggregate routes
  n Generated routes
  n Static routes
	Import all direct and explicitly configured routes into the `inet.0` routing table.	Does not export any routes. The pseudoprotocol cannot export any routes from the routing table because it is not a routing protocol.
		Routing protocols can export these or any routes from the routing table.

RIP	Import all RIP routes learned from configured neighbors into the `inet.0` routing table.	Does not export any routes. To export RIP routes, you must configure an export policy for RIP.
RIPng	Import all RIPng routes learned from configured neighbors into the `inet6.0` routing table.	Does not export any routes. To export RIPng routes, you must configure an export policy for RIPng.

When multiple routes for a destination exist in the routing table, the protocol selects an active route, and that route is placed in the appropriate routing table. For equal-cost routes, the JUNOS software places multiple next hops in the appropriate routing table. When a protocol is exporting routes from the routing table, it exports active routes only (applies to actions specified by both default and user-defined export policies).

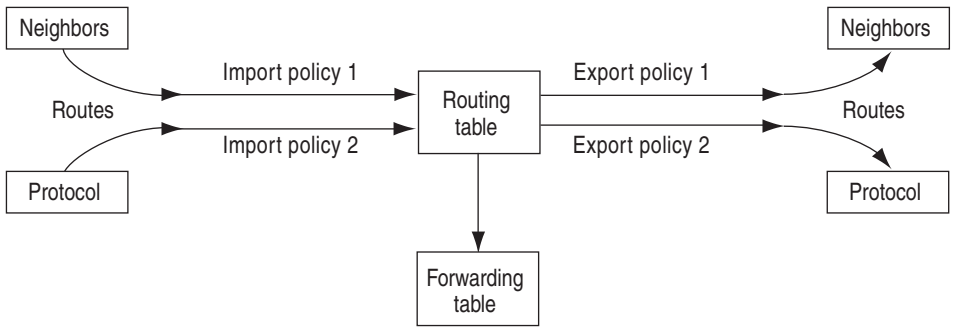
You cannot change the default import policy for the link-state protocols IS-IS and OSPF. As link-state protocols, IS-IS and OSPF exchange routes between systems within an autonomous system (AS). All routers and systems within an AS must share the same link-state database, which includes routes to reachable prefixes and the metrics associated with the prefixes. If an import policy were configured and applied to IS-IS or OSPF, some routes might not be learned or advertised, or the metrics for learned routes might be altered, which would make a consistent link-state database impossible.

The following default actions are taken if one of the following situations arises during policy evaluation:

- If a policy does not specify a match condition, all routes evaluated against the policy match.
- If a match occurs but the policy does not specify an accept, reject, next term, or next policy action, one of the following occurs:
  - The next term, if present, is evaluated.
  - If no other terms are present, the next policy is evaluated.
  - If no other policies are present, the action specified by the default policy is taken.
- If a match does not occur with a term in a policy and subsequent terms in the same policy exist, the next term is evaluated.
- If a match does not occur with any terms in a policy and subsequent policies exist, the next policy is evaluated.
- If a match does not occur by the end of a policy or all policies, the accept or reject action specified by the default policy is taken.

When you configure routing policy, you use *import routing policies* to control which routes routing protocols place in the routing table and *export routing policies* to control which routes a routing protocol advertises from the routing table to its neighbors (see Figure 8.6).

Figure 8.6 *Import and Export Routing Policies*



To define a routing policy, you create a *term* that specifies *match conditions*, which are criteria that a route must match, and *actions*, which is what to do if a route matches the conditions. The actions can specify whether to accept or reject the route, control how a series of

policies is evaluated, and manipulate the characteristics associated with a route. You define match conditions and actions within a *term*. After defining a routing policy, you then apply it to a routing protocol or to the forwarding table.

Routing policy match conditions fall into two categories: standard and extended (see Table 8.5). In general, the standard match conditions include criteria that are defined within a routing policy and are less complex than the extended match conditions, and the extended match conditions include criteria that are defined separately from the routing policy (AS path regular expressions, communities, and prefix lists) and are more complex than standard match conditions. Some match conditions, including communities, prefix lists, and AS path regular expressions, are defined separately from the routing policy and are given names. You then reference the name of the match condition in the definition of the routing policy itself. Named match conditions allow you to reuse match conditions in other routing policies and read configurations that include complex match conditions with more ease.

Table 8.5 *Match Conditions*

Match Condition	Category	When to Use	Notes
AS path regular expression—Combination of AS numbers and regular expression operators	Extended	(BGP only) Match a route based on its AS path.	—
Community—Group of destinations that share a common property (community information is included as a path attribute in BGP update messages)	Extended	Match a group of destinations that share a common property. Use a routing policy to define a community that specifies a group of destinations you want to match and one or more actions that you want taken on this community.	Actions can be performed on the entire group. You can create multiple communities associated with one particular destination. You can create match conditions using regular expressions.
Prefix list—Named list of IP addresses	Extended	Match a route based on prefix information. You can specify an exact match of a particular route only.	You can specify a common action only for all prefixes in the list.
Route list—List of destination prefixes	Extended	Match a route based on prefix information. You can specify an exact match of a particular route or a less precise match.	You can specify an action for each prefix in the route list or a common action for all prefixes in the route list.

Table 8.5 *Match Conditions*

Match Condition	Category	When to Use	Notes
Standard—Collection of criteria that can match a route	Standard	Match a route based on one of the following criteria: area ID, color, external route, family, instance (routing), interface name, level number, local preference, metric, neighbor address, next-hop address, origin, preference, protocol, routing table name, or tag. For the protocol criterion, you can specify one of the following: BGP, direct, DVMRP, IS-IS, local, MPLS, OSPF, PIM dense mode, PIM sparse mode, RIP, RIPng, static, and aggregate.	—
Subroutine—Routing policy that is called repeatedly from another routing policy.	Extended	Use an effective routing policy in other routing policies.	The subroutine action influences but does not necessarily determine the final action.

An *action* is what the policy framework software performs if a route matches all criteria defined in a match condition. You can configure one or more actions in a term. The policy framework software supports flow control actions, which affect whether to accept or reject the route or whether to evaluate the next term or routing policy, actions that manipulate route characteristics, and trace action, which logs route matches.

A *term* is a named structure in which match conditions and actions are defined. You can define one or more terms. In general, the policy framework software compares a route against the match conditions in the first term in the first routing policy, then goes on to the next term and the next policy if present, and so on until an explicitly configured or default action of accept or reject is taken. Therefore, the order in which you arrange terms in a policy is relevant. The order of match conditions in a term is not relevant because a route must match all match conditions in a term for an action to be taken.

After defining a routing policy, you can apply it to routing protocols, to a pseudoprotocol (explicitly created routes, which include aggregate and generated routes), and to the forwarding table. Table 8.6 summarizes the import and export policy support for each routing protocol. You can apply an import policy to aggregate and generated routes, but you cannot apply an export policy to these routes. These routes cannot be exported from the routing table to the pseudoprotocol, because this protocol is not a real routing protocol. However, aggregate and generated routes can be exported from the routing table to routing protocols.

Table 8.6 *Apply Routing Policies to Protocols*

Protocol	Import Policy	Export Policy	Supported Levels
BGP	Yes	Yes	Import: global, group, peer Export: global, group, peer
DVMRP	Yes	Yes	Global
IS-IS	No	Yes	Export: global
LDP	Yes	Yes	Global
MPLS	No	No	—
OSPF	No	Yes	Export: global
PIM dense mode	Yes	Yes	Global
PIM sparse mode	Yes	Yes	Global
Pseudoprotocol—Explicitly configured routes, including aggregate routes and generated routes	Yes	No	Import: global
RIP and RIPng	Yes	Yes	Import: global, neighbor Export: group

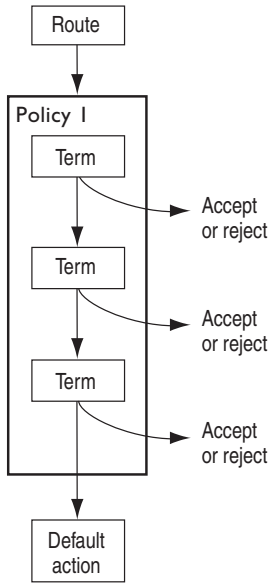
For BGP only, you can also apply import and export policies at group and peer levels as well as at the global level. A peer import or export policy overrides a group import or export policy. A group import or export policy overrides a global import or export policy.

For RIP and RIPng only, you can apply import policies at the global and neighbor levels and export policies at a group level.

You can apply export policies to routes being exported from the routing table into the forwarding table for per-packet load balancing and CoS.

Figure 8.7 shows how a single routing policy is evaluated. This routing policy consists of multiple terms. Each term consists of match conditions and actions to apply to matching routes. Each route is evaluated against the policy as follows:

Figure 8.7 *Routing Policy Evaluation*



1. The route is evaluated against the first term. If it matches, the specified action is taken. If the action is to accept or reject the route, that action is taken and the evaluation of the route ends. If the next term action is specified, if an action is not specified, or if the route does not match, the evaluation continues as described in Step 2. If the next policy action is specified, any accept or reject action specified in this term is skipped, all remaining terms in this policy are skipped, all other actions are taken, and the evaluation continues as described in Step 3.

2. The route is evaluated against the second term. If it matches, the specified action is taken. If the action is to accept or reject the route, that action is taken and the evaluation of the route ends. If the next term action is specified, if an action is not specified, or if the route does not match, the evaluation continues in a similar manner against the last term. If the next policy action is specified, any accept or reject action specified in this term is skipped, all remaining terms in this policy are skipped, all other actions are taken, and the evaluation continues as described in Step 3.
3. If the route matches no terms in the routing policy or the next policy action is specified, the accept or reject action specified by the default policy is taken.

Figure 8.8 shows how a chain of routing policies is evaluated. These routing policies consist of multiple terms. Each term consists of match conditions and actions to apply to matching routes. Each route is evaluated against the policies as follows:

1. The route is evaluated against the first term in the first routing policy. If it matches, the specified action is taken. If the action is to accept or reject the route, that action is taken and the evaluation of the route ends. If the next term action is specified, if an action is not specified, or if the route does not match, the evaluation continues as described in Step 2. If the next policy action is specified, any accept or reject action specified in this term is skipped, all remaining terms in this policy are skipped, all other actions are taken, and the evaluation continues as described in Step 3.
2. The route is evaluated against the second term in the first routing policy. If it matches, the specified action is taken. If the action is to accept or reject the route, that action is taken and the evaluation of the route ends. If the next term action is specified, if an action is not specified, or if the route does not match, the evaluation continues in a similar manner against the last term in the first routing policy. If the next policy action is specified, any accept or reject action specified in this term is skipped, all remaining terms in this policy are skipped, all other actions are taken, and the evaluation continues as described in Step 3.

3. If the route does not match a term or matches a term with a next policy action in the first routing policy, it is evaluated against the first term in the second routing policy.
4. The evaluation continues until the route matches a term with an accept or reject action defined or until there are no more routing policies to evaluate. If there are no more routing policies, the accept or reject action specified by the default policy is taken.

Figure 8.8 *Routing Policy Chain Evaluation*

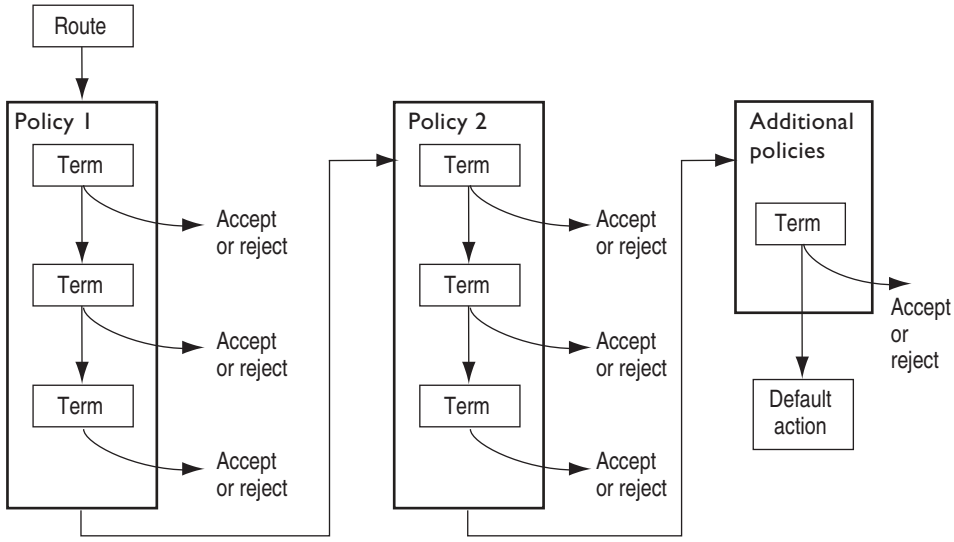


Figure 8.9 on page 325 shows how a subroutine is evaluated. The subroutine is included in the first term of the first routing policy in a chain. Each route is evaluated against the subroutine as follows:

1. The route is evaluated against the first term in the first routing policy. If the route does not match all match conditions specified before the subroutine, the subroutine is skipped and the next term in the routing policy is evaluated (see Step 2). If the route matches all match conditions specified before the subroutine, the route is evaluated against the subroutine. If the route matches the match conditions in any of the subroutine terms, two levels of evaluation occur in the following order:

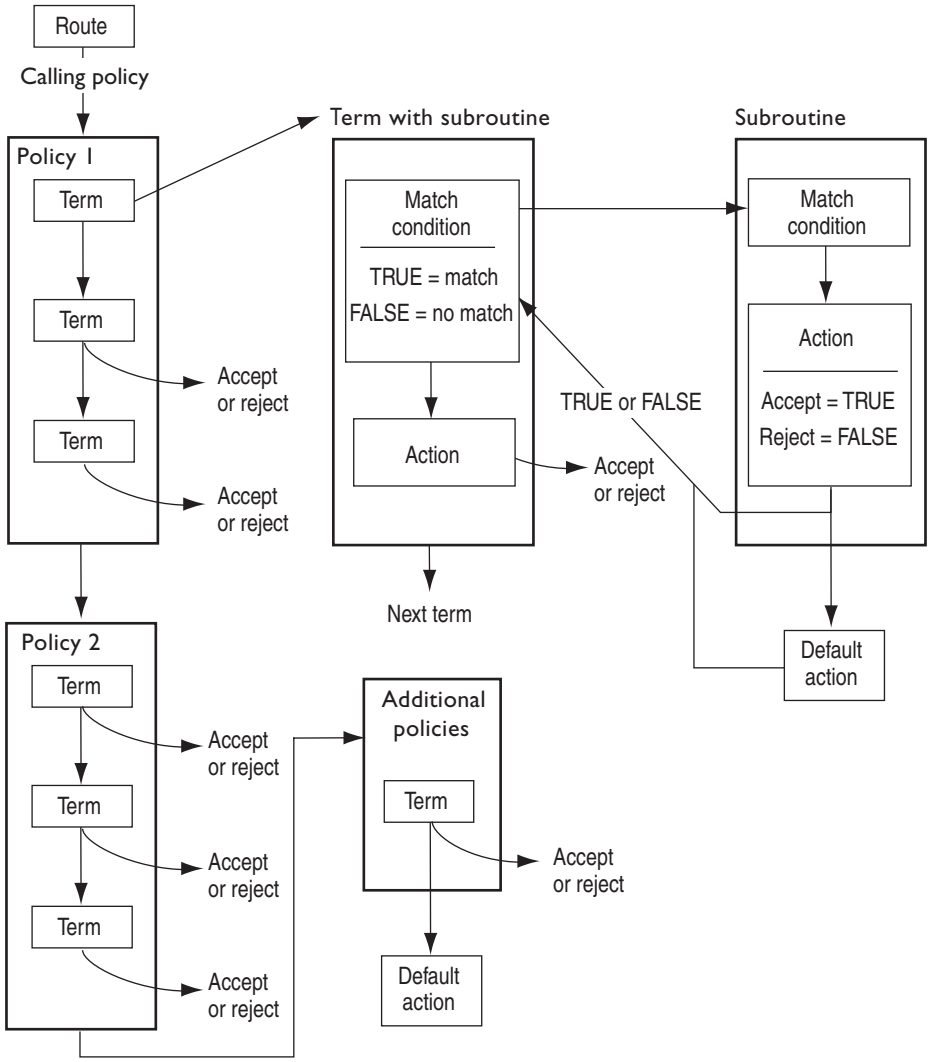
a. The actions in the subroutine term are evaluated. If one of the actions is accept, evaluation of the subroutine ends and a Boolean value of TRUE is returned to the calling policy. If one of the actions is reject, evaluation of the subroutine ends and FALSE is returned to the calling policy. If one of the actions is meant to manipulate route characteristics, the characteristic is changed regardless of whether accept, reject, or neither action is specified.

   If the subroutine does not specify the accept or reject actions, it uses the accept or reject action specified by the default policy and the values of TRUE or FALSE are returned to the calling policy as described above.

b. The calling policy's subroutine match condition is evaluated. During this part of the evaluation, TRUE equals a match and FALSE equals no match. If the subroutine returned TRUE to the calling policy, then the evaluation of the calling policy continues. If the subroutine returned FALSE to the calling policy, then the evaluation of the current term ends and the next term is evaluated.

2. The route is evaluated against the second term in the first routing policy. If a term defines multiple match conditions, including a subroutine, and a route does not match a condition specified before the subroutine, the evaluation of the term ends and the subroutine is not called and evaluated. In this situation, an action specified in the subroutine that manipulates a route's characteristics is not implemented. If you specify a policy chain as a subroutine, the entire chain acts as a single subroutine. That is, as with other chains, the action specified by the default policy is taken only when the entire chain does not accept or reject a route.

Figure 8.9 *Routing Policy Subroutine Evaluation*



Configuring Routing Policy

To configure routing policy, you first define the policy, then apply it to a routing protocol or the forwarding table.

Defining Routing Policies

To define a routing policy, include the `policy-statement` statement. Each routing policy name must be unique within a configuration.

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      match-conditions;
      policy subroutine-policy-name;
      prefix-list name;
      route-filter destination-prefix match-type <actions>;
      source-address-filter destination-prefix match-type
        <actions>;
    }
    to {
      match-conditions;
      policy subroutine-policy-name;
    }
    then actions;
    prefix-list name {
    ip-addresses;
  }
}
```

Each policy term can consist of two statements, `from` and `to`, that define match conditions. In the `from` statement, define the criteria that an incoming route must match. You can specify one or more match conditions. If you specify more than one, they all must match the route for a match to occur. The `from` statement is optional. If you omit it and the `to` statement, all routes are considered to match. In export policies, omitting the `from` statement in a routing policy term might lead to unexpected results. In the `to` statement, define the criteria that an outgoing route must match. You can specify one or more match conditions. If you specify more than one, they all must match the route for a match to occur. You can specify most of the same match conditions in the `to` statement as in the `from` statement. In most cases, specifying a match condition in the `to` statement produces the same

result as specifying the same match condition in the `from` statement. The `to` statement is optional. If you omit both it and the `from` statement, all routes are considered to match.

All conditions in the `from` and `to` statements must match for the action to be taken. The match conditions are effectively a logical AND operation. Table 8.7 describes the match conditions for incoming and outgoing routes. This table indicates whether the match condition is standard or extended. In general, the extended match conditions include criteria that are defined separately from the routing policy (AS path regular expressions, communities, and prefix lists) and are more complex than standard match conditions.

Table 8.7 *Routing Policy Match Conditions*

Match Condition	Match Condition Category	`from` Statement Description `to` Statement Description
`area` *`area-id`*	Standard	(OSPF only) Area identifier. In a `from` statement used with an `export` policy, match a route learned from the specified OSPF area when exporting OSPF routes into other protocols.
`as-path` *`name`*	Extended	(BGP only) Name of an AS path regular expression.
`color` *`preference`* `color2` *`preference`*	Standard	Color value, for preference values (`color` and `color2`) that are finer-grained than those specified in the `preference` and `preference2` match conditions. The color value can be a number in the range 0 through 4,294,967,295 (2^{32} –1), with a lower number indicating a more-preferred route. For more information about preference values, see Table 9.1, "Default Route Preference Values," on page 379.
`community [`*`names`*`]`	Extended	Name of one or more communities. If you list more than one name, only one name needs to match for a match to occur. The community matching is effectively a logical OR operation.
`external [type` *`metric-type`*`]`	Standard	(OSPF only) External routes, including routes exported from one level to another. `type` is an optional keyword. *`metric`* can either be 1 or 2. When you do not specify `type`, this condition matches all external routes. When you specify `type`, this condition matches only OSPF routes with the specified OSPF metric type.
`family` *`family-name`*	Standard	Name of an address family. *`family-name`* can be either `inet` or `inet6`. Match the address family (IPv4 or IPv6) of the route.

Table 8.7 *Routing Policy Match Conditions*

Match Condition	Match Condition Category	`from` Statement Description	`to` Statement Description
`instance` *instance-name*	Standard	Routing instance or instances specified by name. Match a route learned from one of the specified instances.	Routing instance or instances specified by name. Match a route to be advertised over one of the specified instances.
`interface` *interface-name*	Standard	Router interface or interfaces specified by name or IP address. Do not use this qualifier with protocols that are not interface specific, such as IBGP. Match a route learned from one of the specified interfaces. Direct routes match routes configured on the specified interface.	Router interface or interfaces specified by name or IP address. Do not use this qualifier with protocols that are not interface specific, such as IBGP. Match a route to be advertised from one of the specified interfaces.
`level` *level*	Standard	(IS-IS only) IS-IS level. Match a route learned from a specified level.	(IS-IS only) IS-IS level. Match a route to be advertised to a specified level.
`local-preference` *value*	Standard	(BGP only) BGP local preference (`LOCAL_PREF`) attribute. The value can be a number in the range 0 through 4,294,967,295 (2^{32} –1).	
`metric` *metric* `metric2` *metric* `metric3` *metric* `metric4` *metric*	Standard	Metric value. You can specify up to four metric values, starting with `metric` (for the first metric value) and continuing with `metric2`, `metric3`, and `metric4`. (BGP only) `metric` corresponds to the MED, and `metric2` corresponds to the IGP metric if the BGP next hop runs back through another route.	
`neighbor` *address*	Standard	Neighbor (peer) address or addresses. For BGP, the address can be a directly connected or indirectly connected peer. For all other protocols, the address is the neighbor from which the advertisement is received.	Neighbor (peer) address or addresses. For BGP import policies, specifying `to neighbor` produces the same result as specifying `from neighbor`. For BGP export policies, specifying the `neighbor` match condition has no effect and is ignored. For all other protocols, the `to` statement matches the neighbor to which the advertisement is sent.
`next-hop` *address*	Standard	Next-hop address or addresses specified in the routing information for a particular route.	

Table 8.7 *Routing Policy Match Conditions*

Match Condition	Match Condition Category	`from` Statement Description	`to` Statement Description
`origin` *value*	Standard	(BGP only) BGP origin attribute, which is the origin of the AS path information. The value can be one of the following: n `egp`—Path information originated in another AS. n `igp`—Path information originated within the local AS. n `incomplete`—Path information was learned by some other means.	
`policy [` *policy-name* `]`	Extended	Name of a policy to evaluate as a subroutine.	
`preference` *preference* `preference2` *preference*	Standard	Preference value. You can specify a primary preference value (`preference`) and a secondary preference value (`preference2`). The value can be a number in the range 0 through 4,294,967,295 (2^{32} –1), with a lower number indicating a more-preferred route. To specify even finer-grained preference values, see the `color` and `color2` match conditions in this table. For more information about preference values, see Table 9.1, "Default Route Preference Values," on page 379.	
`prefix-list` *name* *ip-addresses*	Extended	Named list of IP addresses. You can specify an exact match with incoming routes.	You cannot specify this match condition.
`protocol` *protocol*	Standard	Name of the protocol from which the route was learned or to which the route is being advertised. It can be one of the following: `aggregate`, `bgp`, `direct`, `dvmrp`, `isis`, `local`, `ospf`, `pim-dense`, `pim-sparse`, `rip`, `ripng`, or `static`.	
`rib` *routing-table*	Standard	Name of a routing table. It can be one of the following: n `inet.0`—Unicast IPv4 routes. n *instance-name*`.inet.0`—Unicast IPv4 routes for a particular routing instance. n `inet.1`—Multicast IPv4 routes. n `inet.2`—Unicast IPv4 routes for multicast reverse-path forwarding (RPF) lookup. n `inet.3`—MPLS routes. n `mpls.0`—MPLS routes for label-switched path (LSP) next hops. n `inet6.0`—Unicast IPv6 routes.	

Table 8.7 *Routing Policy Match Conditions*

Match Condition	Match Condition Category	`from` Statement Description	`to` Statement Description
`route-filter` *`destination-prefix match-type <actions>`*	Extended	List of destination prefixes. When specifying a destination prefix, you can specify an exact match with a specific route or a less precise match using match types. You can configure either a common action that applies to the entire list or an action associated with each prefix.	You cannot specify this match condition.
`source-address-filter` *`destination-prefix match-type <actions>`*	Extended	List of multicast source addresses. When specifying a source address, you can specify an exact match with a specific route or a less precise match using match types. You can configure either a common action that applies to the entire list or an action associated with each prefix.	You cannot specify this match condition.
`tag` *`string`* `tag2` *`string`*	Standard	You can specify two tag strings: `tag` (for the first string) and `tag2`. These values are local to the router and can be set on configured routes or by using an import routing policy.	

Each policy term can include a `then` statement, which defines the actions to take if a route matches all the conditions in the `from` and `to` statements. If a term does not have `from` and `to` statements, all routes are considered to match and the actions apply to them all. The `then` statement is optional. You can specify one or more actions. There are three types of actions: flow control actions (see Table 8.8), which affect whether to accept or reject the route and whether to evaluate the next term or routing policy, actions that manipulate route characteristics (see Table 8.9), and trace action, which logs route matches.

If you do not include a `then` statement, the next term in the routing policy, if one is present, is evaluated; if there are no more terms in the routing policy, the next routing policy, if one is present, is evaluated; or if there are no more terms or routing policies, the accept or reject action specified by the default policy is taken.

Table 8.8 *Flow Control Actions*

Flow Control Action	Description
`accept`	Accept the route and propagate it. After a route is accepted, no other terms in the routing policy and no other routing policies are evaluated.
`reject`	Reject the route and do not propagate it. After a route is rejected, no other terms in the routing policy and no other routing policies are evaluated.
`next term`	Skip to and evaluate the next term in the same routing policy. Any accept or reject action specified in the `then` statement is skipped. Any actions in the `then` statement that manipulate route characteristics are applied to the route. `next term` is the default control action if a match occurs and if you do not specify a flow control action.
`next policy`	Skip to and evaluate the next routing policy. Any accept or reject action specified in the `then` statement is skipped. Any actions in the `then` statement that manipulate route characteristics are applied to the route. `next policy` is the default control action if a match occurs, if you do not specify a flow control action, and if there are no further terms in the current routing policy.

Table 8.9 *Actions That Manipulate Route Characteristics*

Action	Description
`as-path-prepend` *`as-path`*	(BGP only) Affix one or more AS numbers at the beginning of the AS path. To specify more than one AS number, include the numbers in quotation marks. The AS numbers are added after the local AS number has been added to the path. This action adds AS numbers to AS sequences only, not AS sets. If the existing AS path begins with a confederation sequence or set, the affixed AS numbers are placed within a confederation sequence. Otherwise, the affixed AS numbers are placed with a nonconfederation sequence.
`as-path-expand last-as count` *`n`*	(BGP only) Extract the last AS number in the existing AS path and affix that AS number to the beginning of the AS path *n* times, where *n* is from 1 through 32. The AS number is added before the local AS number has been added to the path. This action adds AS numbers to AS sequences only. AS sets are ignored. If the existing AS path begins with a confederation sequence or set, the affixed AS numbers are placed within a confederation sequence. Otherwise, the affixed AS numbers are placed with a nonconfederation sequence. This option is typically used in EBGP export policies.
`class` *`class-name`*	(CoS only) Apply parameters to routes installed into the routing table. For more information about CoS, see Chapter 6,"Interfaces and Class of Service," on page 185.

Table 8.9 *Actions That Manipulate Route Characteristics*

Action	Description
`color` *preference* `color2` *preference*	Set the preference value to a specific value. The `color` and `color2` preference values are even finer-grained than those specified in the `preference` and `preference2` actions. The color value can be a number in the range 0 through 4,294,967,295 (2^{32} –1), with a lower number indicating a more-preferred route. If you set the preference with the `color` action, the value is internal to the JUNOS software and is not transitive. For more information about preference values, see Table 9.1, "Default Route Preference Values," on page 379.
`color (add | subtract) number` `color2 (add | subtract) number`	Change the color preference value by the specified amount. If an addition operation results in a value that is greater than 4,294,967,295 (2^{32} –1), the value is set to 2^{32} –1. If a subtraction operation results in a value less than 0, the value is set to 0. If no attribute value is set before the addition or subtraction operation, the attribute value defaults to 0. If you perform an addition to an attribute with a value of 0, that number becomes the constant value.
`community (+ | add) [ names ]`	(BGP only) Add communities to the set of communities in the route.
`community (- | delete) [ names ]`	(BGP only) Delete communities from the set of communities in the route.
`community (= | set) [ names ]`	(BGP only) Set the communities in the route, replacing any communities that were in the route.
`damping name`	(BGP only) Apply route-damping parameters to the route. These parameters override the default damping parameters. This action is useful only in an import policy, because the damping parameters affect the state of routes in the routing table. To apply damping parameters, you must enable BGP flap damping as described in "Configuring Route Flap Damping," on page 406, and you must create a named list of parameters.
`destination-class` `destination-class-name`	Maintain packet counts for a route passing through your network based on the ingress and egress points. To configure, group destination prefixes by configuring a routing policy. Enable packet counting on one or more interfaces by including the `destination-class-usage` statement at the `[edit interfaces interface-name unit logical-unit-number family inet]` hierarchy level, then apply that routing policy to the forwarding table with the corresponding destination class.
`external type metric`	Set the external metric type for routes exported by OSPF. You must specify the keyword `type`.
`install-nexthop lsp lsp-name`	Choose which, among a set of equal-cost label-switched path (LSP) next hops, are installed in the forwarding table. To choose, use the export policy for the forwarding table to specify the LSP next hop to be used for the desired routes.
`load-balance per-packet`	(For export to the forwarding table only) Install all next-hop addresses into the forwarding table and have the forwarding table perform per-packet load balancing.
`local-preference value`	(BGP only) Set the BGP local preference (`LOCAL_PREF`) attribute. The preference value can be a number in the range 0 through 4,294,967,295.

Table 8.9 *Actions That Manipulate Route Characteristics*

Action	Description
`local-preference (add | subtract) number`	Change the local preference value by the specified amount. If an addition operation results in a value that is greater than 4,294,967,295 (2^{32} –1), the value is set to 2^{32} –1. If a subtraction operation results in a value less than 0, the value is set to 0. If no attribute value is set before the addition or subtraction operation, the attribute value defaults to 0. If you perform an addition to an attribute with a value of 0, that number becomes the constant value. For BGP, if the attribute value is not known, the local preference attribute value is initialized to 100 before the routing policy is applied.
`metric metric metric2 metric metric3 metric metric4 metric`	Set the metric. You can specify up to four metric values, starting with `metric` (for the first metric value) and continuing with `metric2`, `metric3`, and `metric4`. (BGP only) `metric` corresponds to the multiple exit discriminator (MED), and `metric2` corresponds to the IGP metric if the BGP next hop recourses through another router.
`metric (add | subtract) number metric2 (add | subtract) number metric3 (add | subtract) number metric4 (add | subtract) number`	Change the metric value by the specified amount. If an addition operation results in a value that is greater than 4,294,967,295 (2^{32} –1), the value is set to 2^{32} –1. If a subtraction operation results in a value less than 0, the value is set to 0. If no attribute value is set before the addition or subtraction operation, the attribute value defaults to 0. If you perform an addition to an attribute with a value of 0, that number becomes the constant value.
`metric (igp | minimum-igp) site-offset`	(BGP only) Change the MED value by the specified negative or positive offset. This action is useful only in an EBGP export policy.
`next-hop (address | peer-address)`	Set the next hop. When the advertising protocol is BGP, you can set the next hop only when any third-party next hop can be advertised; that is, when using IBGP or EBGP confederations. If you specify *`address`* as `self`, the next-hop address is replaced by one of the local router's addresses. The advertising protocol determines which address to use. When the advertising protocol is BGP, this address is set to the local IP address used for the BGP adjacency. A router cannot install routes with itself as the next hop. If you specify `peer-address`, the next-hop address is replaced by the peer's IP address. This option is only valid in import policies. Primarily used by BGP to enforce using the peer's IP address for advertised routes, this option is only meaningful when the next hop is the advertising router or another directly connected router.
`origin` *`value`*	(BGP only) Set the BGP origin attribute to one of the following values: n `igp`—Path information originated within the local AS. n `egp`—Path information originated in another AS. n `incomplete`—Path information was learned by some other means.

Table 8.9 *Actions That Manipulate Route Characteristics*

Action	Description
`preference` *preference* `preference2` *preference*	Set the preference value. You can specify a primary preference value (`preference`) and a secondary preference value (`preference2`). The preference value can be a number in the range 0 through 4,294,967,295 (2^{32} –1), with a lower number indicating a more-preferred route. To specify even finer-grained preference values, see the `color` and `color2` actions in this table. If you set the preference with the `preference` action, the new preference remains associated with the route. The new preference is internal to the JUNOS software and is not transitive. For more information about preference values, see the Table 9.1, "Default Route Preference Values," on page 379.
`preference (add | subtract)` *number* `preference2 (add | subtract)` *number*	Change the preference value by the specified amount. If an addition operation results in a value that is greater than 4,294,967,295 (2^{32} –1), the value is set to 2^{32} –1. If a subtraction operation results in a value less than 0, the value is set to 0. If no attribute value is set before the addition or subtraction operation, the attribute value defaults to 0. If you perform an addition to an attribute with a value of 0, that number becomes the constant value.
`tag` *tag* `tag2` *tag*	You can specify two tag strings: `tag` (for the first string) and `tag2`. These values are local to the router. For OSPF only, the `tag` and `tag2` actions set the 32-bit tag field in OSPF external LSA packets.
`tag (add | subtract)` *number* `tag2 (add | subtract)` *number*	Change the tag value by the specified amount. If an addition operation results in a value that is greater than 4,294,967,295 (2^{32} –1), the value is set to 2^{32} –1. If a subtraction operation results in a value less than 0, the value is set to 0. If no attribute value is set before the addition or subtraction operation, the attribute value defaults to 0. If you perform an addition to an attribute with a value of 0, that number becomes the constant value.

Applying Routing Policies

For a routing policy to take effect, you must apply it to either a routing protocol or the forwarding table. To apply a routing policy to a routing protocol, include the `import` and `export` statements:

```
[edit protocols protocol-name]
import policy-name;
export policy-name;
```

In the `import` statement, list the name of the routing policy to be evaluated when routes are imported into the routing table from the routing protocol. In the `export` statement, list the name of the routing policy to be evaluated when routes are being exported from the routing table into a dynamic routing protocol. Only active routes are exported from the routing table.

To apply multiple routing policies (chains) to a routing protocol, include the `import` and `export` statements at the `[edit protocols` *`protocol-name`*`]` hierarchy level:

```
[edit protocols protocol-name]
import [ policy-name policy-name ... ]
export [ policy-name policy-name ... ]
```

In the `import` statement, list the names of multiple routing policies to be evaluated when routes are imported into the routing table from the routing protocol. In the `export` statement, list the names of multiple routing policies to be evaluated when routes are being exported from the routing table into a dynamic routing protocol. Only active routes are exported from the routing table.

The policy framework software evaluates the routing policies sequentially, from left to right. If an action specified in one of the policies manipulates a route characteristic, the policy framework software carries the new route characteristic forward during the evaluation of the remaining policies. For example, if the action specified in the first policy of a chain sets a route's metric to 500, this route matches the criterion of `metric 500` defined in the next policy.

The policy framework software can evaluate routing policies using *policy expressions*, which use Boolean logical operators with policies. The logical operators establish rules by which the policies are evaluated. During evaluation of a routing policy in a policy expression, the policy actions of accept, reject, or next policy are converted to the value of TRUE or FALSE. This value is then evaluated against the logic of the specified logical operator to produce output of either TRUE or FALSE. The output is then converted back to a flow control action of accept, reject, or next policy. The result of the policy expression is applied in the same manner as it would be applied to a single policy; that is, the route is accepted or rejected and the evaluation ends, or the next policy is evaluated. Table 8.10 summarizes the policy actions and their corresponding TRUE and FALSE values and flow control action values. Table 8.11 describes the logical operators. You can place a policy expression anywhere in the `import` or `export` statements and in the `from policy` statement, enclosing the expression in parentheses.

Table 8.10 *Policy Action Conversion Values*

Policy Action	Conversion Value	Flow Control Action Conversion Value
accept	TRUE	accept
reject	FALSE	reject
next policy	TRUE	next policy

Table 8.11 *Policy Expression Logical Operators*

Logical Operator	Policy Expression Logic	How Logical Operator Affects Policy Expression Evaluation
&& (Logical AND)	Logical AND requires that all values must be TRUE to produce output of TRUE. Routing policy value of TRUE and TRUE produces output of TRUE. Value of TRUE and FALSE produces output of FALSE. Value of FALSE and FALSE produces output of FALSE.	If the first routing policy returns the value of TRUE, the next policy is evaluated. If the first policy returns the value of FALSE, the evaluation of the expression ends and subsequent policies in the expression are not evaluated.
|| (Logical OR)	Logical OR requires that at least one value must be TRUE to produce output of TRUE. Routing policy value of TRUE and FALSE produces output of TRUE. Value of TRUE and TRUE produces output of TRUE. Value of FALSE and FALSE produces output of FALSE.	If the first routing policy returns the value of TRUE, the evaluation of the expression ends and subsequent policies in the expression are not evaluated. If the first policy returns the value of FALSE, the next policy is evaluated.
! (Logical NOT)	Logical NOT reverses value of TRUE to FALSE and of FALSE to TRUE. Also reverses the actions of accept and next policy to reject, and reject to accept.	If used with the logical AND operator and the first routing policy value of FALSE is reversed to TRUE, the next policy is evaluated. If the value of TRUE is reversed to FALSE, the evaluation of the expression ends and subsequent policies in the expression are not evaluated. If used with the logical OR operator and the first routing policy value of FALSE is reversed to TRUE, the evaluation of the expression ends and subsequent policies in the expression are not evaluated. If the value of TRUE is reversed to FALSE, the next policy is evaluated. If used with a policy and the flow control action is accept or next policy, these actions are reversed to reject. If the flow control action is `reject`, this action is reversed to `accept`.

The policy framework software evaluates a policy expression as follows:

1. The software evaluates a route against the first routing policy in a policy expression and converts the specified or default action to a value of TRUE or FALSE.
2. The software takes the value of TRUE or FALSE and evaluates it against the logical operator used in the policy expression. Based on the logical operator used, the software determines whether to evaluate the next routing policy, if one is present.

   The policy framework software uses a method of shortcut evaluation. When a result is certain, the software stops evaluating subsequent routing policies in the policy expression. For example, if the policy expression specifies logical AND and the evaluation of the first routing policy returns the value of FALSE, the software determines that the output will be FALSE no matter what the value of the unevaluated routing policies are. Therefore, the software does not evaluate the subsequent routing policies in this policy expression.

3. The software performs the same tasks described in Steps 1 and 2 for each subsequent routing policy in the policy expression, if they are present and if the software has determined that it is appropriate to evaluate them.
4. After evaluating the last routing policy, if it is appropriate, the software evaluates the value of TRUE or FALSE obtained from each routing policy evaluation. Based on the logical operator used, it calculates an output of TRUE or FALSE.
5. The software converts the output of TRUE or FALSE back to an action, and the action is performed.

   In cases where each policy in the expression returned a value of TRUE, the software converts the output of TRUE back to the flow control action specified in the last policy. For example, if the policy expression `(policy1 && policy2)` is specified and `policy1` specifies `accept` and `policy2` specifies `next term`, the `next term` action is performed.

If an action specified in one of the policies manipulates a route characteristic, the policy framework software carries the new route characteristic forward during the evaluation of the remaining policies. For

example, if the action specified in the first policy of a policy expression sets a route's metric to 500, this route matches the criteria of `metric 500` defined in the next policy. However, if a route characteristic manipulation action is specified in a policy located in the middle or the end of a policy expression, it is possible, because of the shortcut evaluation, that the policy is never evaluated and the manipulation of the route characteristic never occurs.

Applying Routing Policies to the Forwarding Table

To apply an export routing policy to the forwarding table for per-packet load balancing and CoS, include the `forwarding-table` statement:

```
[edit routing-options]
forwarding-table {
  export [ policy-name ];
}
```

Configuring AS Path Regular Expressions

A BGP *AS path* is a path to a destination. To define a match condition based on all or portions of the AS path, first define the AS path in the `as-path` statement:

```
[edit policy-options]
as-path name regular-expression;
```

Then, name the AS path regular expression in a routing policy, including the `as-path` match condition in the `from` statement:

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      as-path names;
    }
  }
}
```

The regular expression, which is used to match all or portions of the AS path, consists of two components, which you specify in the following format:

```
term <operator>
```

The term identifies an AS. You can specify it in one of the following ways:

- AS number—The entire AS number composes one *term*. You cannot reference individual characters within an AS number, which differs from regular expressions as defined in POSIX 1003.2.
- Wildcard character—Matches any single AS number. The wildcard character is a period (.). You can specify multiple wildcard characters.
- AS path—A single AS number or a group of AS numbers enclosed in parentheses. Grouping the regular expression in this way allows you to perform a common operation on the group as a whole and to give the group precedence. The grouped path can itself include operators.

You can specify one or more term–operator pairs in a single regular expression.

The optional operator defines pattern matching operations to apply to the term. You define the operations using regular expressions. Table 8.12 lists the regular expression operators supported for AS paths. You place operators immediately after *term* with no intervening space, except for the pipe (|) and dash (–) operators, which you place between two terms, and parentheses, with which you enclose terms.

Table 8.12 *AS Path and Community Attribute and Regular Expressions Operators*

Operator	Match
{*m*,*n*}	At least *m* and at most *n* repetitions of *term*. Both *m* and *n* must be positive integers, and *m* must be smaller than *n*.
{*m*}	Exactly *m* repetitions of *term*. *m* must be a positive integer.
{*m*,}	*m* or more repetitions of *term*. *m* must be a positive integer.
*	Zero or more repetitions of *term*. This is equivalent to {0,}.
+	One or more repetitions of *term*. This is equivalent to {1,}.
?	Zero or one repetition of *term*. This is equivalent to {0,1}.
|	One of the two terms on either side of the pipe.
–	Between a starting and ending range, inclusive.
^	Character at the beginning of an AS path regular expression. This character is added implicitly; therefore, the use of it is optional.
$	Character at the end of an AS path regular expression. This character is added implicitly; therefore, its use is optional.

Table 8.12 *AS Path and Community Attribute and Regular Expressions Operators*

Operator	Match
( )	A group of terms that are enclosed in the parentheses. If enclosed in quotation marks with no intervening space ("()"), indicates a null. Intervening space between the parentheses and the terms is ignored.
[ ]	Set of characters. One character from the set can match. To specify the start and end of a range, use a dash (–).

AS path regular expressions implement the extended (modern) regular expressions as defined in POSIX 1003.2 They are identical to the UNIX regular expressions with the following exceptions:

- The basic unit of matching in an AS path regular expression is the AS number, not an individual character.
- A regular expression matches a route only if the AS path in the route exactly matches *regular-expression*. The equivalent UNIX regular expression is ^*regular-expression*$. For example, the AS path regular expression 1234 is equivalent to the UNIX regular expression ^1234$.
- You can specify a regular expression using wildcard operators.

Configuring Communities

A *BGP community* is a group of destinations that share a common property. Community information is included as a path attribute in BGP update messages and identifies community members and allows you to perform actions on a group without having to elaborate on each member. You can define match conditions based on a BGP community, which this section describes. You can assign community tags to non-BGP routes through configuration (for static, aggregate, or generated routes) or an import routing policy. These tags can then be matched when BGP exports the routes.

To create a named community and define the community members, include the community statement:

```
[edit policy-options]
community name members [ community-ids ];
```

Then, name the community in a routing policy, including the `community` condition in the `from` statement. If you include the names of multiple communities, only one community need match for a match to occur.

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      community names;
    }
  }
}
```

name identifies the community or communities. It can contain letters, numbers, and hyphens (-), and can be up to 255 characters long. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

Specify the community identifier in the following format:

```
as-number:community-value
```

The AS number of the community member can be a value from 1 through 65,534. The community value can be a number from 0 through 65,535. You can specify these numbers in one of the following ways:

- AS number.
- Asterisk (*)—A wildcard character that matches all AS numbers. (In the definition of the community attribute, the asterisk also functions as described in Table 8.12.)
- Period (.)—A wildcard character that matches any single digit in an AS number.
- Group of numbers—A single AS or community number or a group of numbers enclosed in parentheses. Grouping the numbers in this way allows you to perform a common operation on the group as a whole and to give the group precedence. The grouped numbers can themselves include regular expression operators.

You also can specify the community identifier as one of the following well-known community names, which are defined in RFC 1997:

- `no-advertise`—Routes in this community name must not be advertised to other BGP peers.
- `no-export`—Routes in this community must not be advertised outside a BGP confederation boundary.
- `no-export-subconfed`—Routes in this community must not be advertised to external BGP peers, including peers in other members' ASs inside a BGP confederation.

When specifying community identifiers, you can use UNIX-style regular expressions to specify the AS number and the member identifier. A regular expression consists of two components, which you specify in the following format:

```
term<operator>
```

term identifies the string to match. *operator* defines pattern-matching operations to apply to the term. Table 8.12 lists the regular expression operators supported for the community attribute. You place an operator immediately after *term* with no intervening space, except for the pipe (|) and dash (–) operators, which you place between two terms, and parentheses, with which you enclose terms.

Community regular expressions are identical to the UNIX regular expressions. Both implement the extended (or modern) regular expressions as defined in POSIX 1003.2. Community regular expressions evaluate the string specified in the *term* on a character-by-character basis. For example, if you specify 1234:5678 as the *term*, the regular expressions see nine discrete characters, including the colon (:), instead of two sets of numbers (1234 and 5678) separated by a colon.

By default, communities are sent to BGP peers. To suppress the advertisement of communities to a neighbor, remove all communities by defining a wildcard set of communities (here, the community is named `wild`) and then specifying the `community delete` action. When the result of an export policy is an empty set of communities, the community attribute is not sent.

```
[edit policy-options]
community wild members "*:*";
policy-statement policy-name {
  term term-name {
    then community delete wild;
  }
}
```

To configure extended communities, define the community and then reference it in the `from` statement:

```
[edit policy-options]
community name members [ community-ids ];
policy-statement policy-name {
  term term-name {
    from {
      community name;
    }
  }
}
```

The community identifier is the type of extended community in the following format:

type:*administrator*:*assigned-number*

type is the type of extended community and can be either a `target`, `origin`, or `domain-id` community. The `target` community identifies the destination to which the route is going. The `origin` community identifies where the route originated. The `domain-id` community identifies the OSPF domain from which the route originated. *administrator* is the administrator. It is either an AS number or an IPv4 address prefix, depending on the type of extended community. *assigned-number* identifies the local provider. Note that regular expressions are not supported for extended communities.

The policy framework software evaluates communities as follows:

- Each route is evaluated against each named community in a routing policy `from` statement. If a route matches one of the named communities in the `from` statement, the evaluation of the current term continues. If a route does not match, the evaluation of the current term ends.
- The route is evaluated against each member of a named community. The evaluation of all members must be successful for the named community evaluation to be successful.

- Each member in a named community is either a literal community value or a regular expression (applies to community attributes only). Each member is evaluated against each community on the route. (Communities are an unordered property on a route. For example, `1:2 3:4` is the same as `3:4 1:2`.) Only one community from the route must match for the member evaluation to be successful.
- Community regular expressions evaluate the string specified in the *term* on a character-by-character basis. For example, if you specify 1234:5678 as the *term*, the regular expressions see nine discrete characters, including the colon (:), instead of two sets of numbers (1234 and 5678) separated by a colon. For example:

```
[edit]
policy-options {
  policy-statement one {
    from {
      community [ comm-one comm-two ];
    }
  }
  community members comm-one [ 1:2 "^4:(5|6)$" ];
  community members comm-two [ 7:8 9:10 ];
}
```

  To match routing policy `one`, the route must match either `comm-one` or `comm-two`. To match `comm-one`, the route must have a community that matches `1:2` and a community that matches `4:5` or `4:6`. To match `comm-two`, the route must have a community that matches `7:8` and a community that matches `9:10`.

Configuring Prefix Lists and Route Lists

A *prefix list* is a named list of IP addresses. You can specify an exact match with incoming routes and apply a common action to all matching prefixes in the list. A *route list* is a collection of destination prefixes on which a common policy action is performed. In a prefix, you can specify an exact match with a particular route or a less precise match.

A prefix list functions similarly to a route list that contains multiple instances of the `exact` match type only. While some functional similarities exist between these two extended match conditions, so do some important differences, which are summarized in Table 8.13.

Table 8.13 *Prefix List and Route List Differences*

Feature	Prefix List	Route Lists
Match types	Does not support match types. The specified prefixes must be matched exactly.	Supports several match types. For more information, see Table 8.14 on page 347.
Action	Can specify action in a `then` statement only. This action is applied to all prefixes in the list.	Can specify action that is applied to a particular prefix in a `route-filter` match condition in a `from` statement or to all prefixes in the list using a `then` statement.

To create a named prefix list, include the `prefix-list` statement, specifying one or more addresses:

```
[edit policy-options]
prefix-list name {
  ip-addresses;
}
```

To include a prefix list in a routing policy, include the `prefix-list` match condition in the `from` statement, specifying the name of the prefix list:

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      prefix-list name;
    }
    then actions;
  }
}
```

During prefix list evaluation, the policy framework software performs a *longest-match lookup*, which means that the software searches for the prefix in the list with the longest length. (Because the policy framework software scans a list looking for the longest prefix, the order in which you specify the prefixes, from top to bottom, does not matter.) The software then compares a route's source address to the longest prefix. If a match occurs, the evaluation of the current term continues. If a match does not occur, the evaluation of the current term ends. If you specify multiple prefixes in the prefix list, only one prefix need match for a match to occur.

Configuring Route Lists

To configure a route list, include one or more `route-filter` or `source-address-filter` options in the `from` statement:

```
[edit policy-options]
policy-statement policy-name {
  term term-name {
    from {
      route-filter destination-prefix match-type
        <actions>;
      source-address-filter destination-prefix match-type
        <actions>;
    }
    then actions;
  }
}
```

The `route-filter` statement is typically used to match prefixes of any type except for multicast source addresses. The `source-address-filter` option is typically used to match multicast source addresses in multiprotocol BGP (MBGP) and Multicast Source Discovery Protocol (MSDP) environments.

`destination-prefix` is the IPv4 or IPv6 prefix specified as *`prefix/prefix-length`*. *`match-type`* is the type of match to apply to the destination prefix (see Table 8.14). *`actions`* is the action to take if the destination prefix matches.

For a list of actions, see Table 8.8 and Table 8.9 on page 331.

If you specify actions in the `route-filter` or `source-address-filter` statement, they are taken immediately when a match occurs, and the `then` statement is not evaluated. If you specify actions in the `then` statement, they are taken when a match occurs and if an action is not specified in the `route-filter` or `source-address-filter` statement.

During route list evaluation, the policy framework software compares each route's source address with the destination prefixes in the route list. The policy framework software first performs a longest-match lookup, which considers the prefix and prefix length only and not the match type. The following sample route list illustrates this point:

```
route-filter 192.168.0.0/14 upto /24;
route-filter 192.168.0.0/15 exact;
```

Here, the longest prefix is `192.168.0.0/15`, which is based on prefix and prefix length only. Then, if the prefix of an incoming route matches the longest prefix, the software then examines the match type

Table 8.14 *Route List Match Types*

Match Type	Match If ...
`exact`	Route shares the same most-significant bits (described by *`prefix-length`*) and *`prefix-length`* is equal to the route's prefix length.
`longer`	Route shares the same most-significant bits (described by *`prefix-length`*) and *`prefix-length`* is greater than the route's prefix length.
`orlonger`	Route shares the same most-significant bits (described by *`prefix-length`*) and *`prefix-length`* is equal to or greater than the route's prefix length.
`prefix-length-range` *`prefix-length2–prefix-length3`*	Route shares the same most-significant bits (described by *`prefix-length`*) and the route's prefix length falls between *`prefix-length2–prefix-length3`*, inclusive. The `upto` and `prefix-length-range` match types are similar in that both specify the most significant bits and provide a range of prefix lengths that can match. The difference is that `upto` allows you to specify an upper limit only for the prefix length range, whereas `prefix-length-range` allows you to specify both lower and upper limits.
`through` *`destination-prefix`*	Matches all the following: n Route shares the same most-significant bits (described by *`prefix-length`*) of the first destination prefix. n Route shares the same most-significant bits (described by *`prefix-length`*) of the second destination prefix for the number of bits in the prefix length. n Number of bits in the route's prefix length is less than or equal to the number of bits in the second prefix. You do not use the `through` match type in most routing policy configurations.
`upto` *`prefix-length2`*	Route shares the same most-significant bits (described by *`prefix-length`*) and the route's prefix length falls between *`prefix-length`* and *`prefix-length2`*.

and action associated with the longest prefix. If a match occurs, the action specified with the prefix is taken. If an action is not specified with the prefix, the action in the `then` statement is taken. If neither action is specified, the software evaluates the next term or routing policy if present or takes the accept or reject action specified by the default policy. If you specify multiple prefixes in the route list, only one prefix need match for a match to occur. If a match does not occur, the software evaluates the next term or routing policy if present or takes the accept or reject action specified by the default policy.

The order in which you specify the prefixes typically does not matter, because the policy framework software scans the route list looking for the longest prefix during evaluation. An exception to this rule is when you use the same destination prefix multiple times in a list. Here, the

order is important, because the list of identical prefixes is scanned from top to bottom and the first match type that matches the route applies. In the following example, different match types are specified for the same prefix. The route `0.0.0.0/0` would be rejected, the route `0.0.0.0/8` would be marked with `next-hop self`, and the route `0.0.0.0/25` would be rejected.

```
route-filter 0.0.0.0/0 upto /7 reject;
route-filter 0.0.0.0/0 upto /24 next-hop self;
route-filter 0.0.0.0/0 orlonger reject;
```

A common problem when defining a route list is including a shorter prefix that you want to match with a longer, similar prefix in the same list. For example, imagine that the prefix `192.168.254.0/24` is compared against the following route list:

```
route-filter 192.168.0.0/16 orlonger;
route-filter 192.168.254.0/23 exact;
```

Because of the longest-match lookup, the prefix `192.168.254.0/23` is determined to be the longest prefix. An exact match does not occur between `192.168.254.0/24` and `192.168.254.0/23 exact`, so the software concludes that the term does not match and goes on to the next term or routing policy if present or takes the accept or reject action specified by the default policy. The shorter prefix `192.168.0.0/16 orlonger` that you wanted to match is inadvertently ignored. One solution is to remove the prefix `192.168.0.0/16 orlonger` from the route list in this term and move it to a previous term in which it is the only prefix or the longest prefix in the list.

Configuring Subroutines

Using a routing policy called from another routing policy as a match condition makes the called policy a *subroutine*. To configure a subroutine, create the subroutine and specify its name using the `policy` match condition in the `from` or `to` statement of another routing policy:

```
[edit policy-options]
policy-statement subroutine-policy-name {
  term term-name {
    from {
      match-conditions;
      route-filter destination-prefix match-type <actions>;
      source-address-filter destination-prefix match-type
        <actions>;
```

```
      prefix-list name;
    }
    to {
      match-conditions;
    }
    then actions;
  }
}
policy-statement policy-name {
  term term-name {
    from {
      policy subroutine-policy-name;

    to {
      policy subroutine-policy-name;
    }
    then actions;
  }
}
```

Do not evaluate a routing policy within itself. If you attempt to do so, no prefixes will ever match the routing policy.

The action specified in a subroutine is used to provide a match condition to the calling policy. If the subroutine specifies an action of accept, the calling policy considers the route to be a match. If the subroutine specifies an action of reject, the calling policy considers the route to not match. If the subroutine specifies an action that is meant to manipulate the route characteristics, the changes are made.

Configuring the AS Path Prepend Action

To *prepend*, or add, one or more AS numbers at the beginning of an AS path, specify the `as-path-prepend` action in the `then` statement. The AS numbers are added after the local AS number has been added to the path. Prepending an AS path makes a shorter AS path look longer and therefore less preferable to BGP.

```
[edit policy-options]
policy-statement name {
  term name {
    from {
      route-filter destination-prefix match-type <actions>;
    }
    then as-path-prepend "as-path";
  }
}
```

Configuring BGP Route Flap Damping

BGP flap damping is defined in RFC 2439, *BGP Route Flap Damping*.

BGP *route flapping* is the situation in which BGP systems send an excessive number of update messages to advertise network reachability information. BGP *flap damping* is a way to reduce the number of update messages sent between BGP peers, thereby reducing the load on these peers without adversely affecting the route convergence time. Flap damping reduces the number of update messages by marking routes as ineligible for selection as the active or preferable route. Doing this leads to some delay, or *suppression*, in the propagation of route information, but the result is increased network stability.

To apply the changes in BGP, see "Configuring Route Flap Damping," on page 406.

To effect changes to the default BGP flap damping values, include the following statements. Table 8.15 describes the damping parameters.

```
[edit policy-options]
damping name {
  disable;
  half-life minutes;
  max-suppress minutes;
  reuse number;
  suppress number;
}
policy-statement name {
  term name {
    from {
      ...
    }
    then damping name;
  }
}
```

Table 8.15 *Damping Parameters*

Damping Parameter	Description	Default	Possible Values
`half-life minutes`	Decay half-life, in minutes	15 minutes	1 through 45 minutes
`max-suppress minutes`	Maximum hold-down time, in minutes	60 minutes	1 through 720 minutes
`reuse number`	Reuse threshold	750 (unitless)	1 through 20,000 (unitless)
`suppress number`	Cutoff (suppression) threshold	3,000 (unitless)	1 through 20,000 (unitless)

Configuring Per-Packet Load Balancing

For the active route, when there are multiple equal-cost paths to the same destination, by default, the JUNOS software chooses in a random fashion one of the next-hop addresses to install into the forwarding table. Whenever the set of next hops for a destination changes in any way, the next-hop address is rechosen, also in a random fashion. You can configure the JUNOS software so that, for the active route, all next-hop addresses for a destination are installed in the forwarding table, a process called *per-packet load balancing*. You can use load balancing to spread traffic across multiple paths between routers.

On routers with the Internet Processor II ASIC, when per-packet load balancing is configured, traffic between routers with multiple paths is divided into individual traffic flows (up to 16 equal-cost load-balanced paths). Packets for each individual flow are kept on a single interface. To recognize individual flows in the transit traffic, the router examines the source IP address, destination IP address, and protocol.

The router recognizes packets that have all these parameters identical, and it ensures that these packets are sent out through the same interface. This prevents problems that might otherwise occur with packets arriving at their destination out of their original sequence.

For information on this action, see Table 8.9 on page 331.

To configure per-packet load balancing, you configure the `load-balance per-packet` action and apply the routing policy to the forwarding table by including the `export` statement at the `[edit routing-options forwarding-table]` hierarchy level.

Configuring Firewall Filters

To configure firewall filters, include the following statements:

```
[edit]
firewall {
  policer policer-name {
    if-exceeding {
      bandwidth-limit rate;
      burst-size-limit bytes;
    }
    then {
      policer-action;
    }
```

```
    }
    filter filter-name {
      accounting-profile name;
      interface-specific;
      policer policer-name {
        if-exceeding {
          bandwidth-limit rate;
          burst-size-limit bytes;
        }
        then {
          policer-action;
        }
      }
      term term-name {
        from {
          match-conditions;
        }
        then {
          actions;
          action-modifiers;
        }
      }
    }
  }
  interfaces {
    interface-name {
      unit logical-unit-number {
        family inet {
          filter {
            input filter-name;
            output filter-name;
          }
        }
      }
    }
  }
```

Policing does not use the filter match conditions. Instead, it uses the `if-exceeding` statement. For more information, see the JUNOS technical documentation: *Routing Policy*.

Firewall filter terms are evaluated in the order in which you specify them in the configuration. To reorder terms, use the configuration mode `insert` command. For example, the command `insert term up before term start` places the term `up` before the term `start`.

In the `then` statement of a firewall filter term, you specify the action to take if the packet matches the conditions in the `from` statement (see Table 8.16 and Table 8.17).

Table 8.16 *Firewall Filter Actions*

Action	Description
`accept`	Accept a packet. This is the default.
`discard`	Discard a packet silently, without sending an ICMP message. Discarded packets are not available for logging or sampling.
`reject <message-type>`	Discard a packet, sending an ICMP destination unreachable message. Rejected packets can be logged or sampled if you configure either of those action modifiers. You can specify one of the following message codes: `administratively-prohibited` (default), `bad-host-tos`, `bad-network-tos`, `host-prohibited`, `host-unknown`, `host-unreachable`, `network-prohibited`, `network-unknown`, `network-unreachable`, `port-unreachable`, `precedence-cutoff`, `precedence-violation`, `protocol-unreachable`, `source-host-isolated`, `source-route-failed`, or `tcp-reset`. If you specify `tcp-reset`, a TCP reset is returned if the packet is a TCP packet. Otherwise, nothing is returned.
`routing-instance routing-instance`	Specify a routing table to which packets are forwarded.

Table 8.17 *Firewall Filter Action Modifiers*

Action Modifier	Description
`count counter-name`	Increment a counter for this filter. The name can contain letters, numbers, and hyphens (-), and can be up to 24 characters long. A counter name is specific to the filter that uses it, so all interfaces that use the same filter count into the same counter.
`forwarding-class class-name`	Specify a particular forwarding class.
`ipsec-sa sa-name`	Specify an IPSec security association for the packet. Used with the `source-address` and `destination-address` match conditions.
`log`	Log the packet's header information in the Routing Engine. You can access this information from the CLI, but it is not available from network management.
`loss-priority priority`	Set the packet loss priority (PLP) to `any`, `low`, or `high`.
`policer policer-name`	Apply rate limits to the traffic using the named policer.
`sample`	Sample the traffic on the interface. Use this modifier only when traffic sampling is enabled.
`syslog`	Log an alert for this packet. The log can be sent to a server for storage and analysis.

When a firewall filter consists of a single term, the filter is evaluated as follows:

- If the packet matches all the conditions, the action in the `then` statement is taken.
- If the packet does not match all the conditions, it is discarded.

When a firewall filter consists of more than one term, the filter is evaluated sequentially:

- The packet is evaluated against the conditions in the `from` statement in the first term.
- If the packet matches, the action in the `then` statement is taken and the evaluation ends. Subsequent terms in the firewall filter are not evaluated.
- If the packet does not match, it is evaluated against the conditions in the `from` statement in the second term.

  This process continues until either the packet matches the `from` conditions in one of the subsequent terms or there are no more terms.

- If a packet passes through all the terms in the filter without matching any of them, it is discarded.

If a term does not contain a `from` statement, the packet is considered to match and the action in the term's `then` statement is taken. If a term does not contain a `then` statement or if you do not configure an action in the `then` statement, and if the packet matches the conditions in the term's `from` statement, the packet is accepted.

Each firewall filter has an implicit discard action at the end of the filter, which is equivalent to the following explicit filter term. Therefore, if a packet matches none of the terms in the filter, it is discarded.

```
term implicit-rule {
  then discard;
}
```

In the `from` statement in the `firewall filter` term, specify conditions that the packet must match for the action in the `then` statement to be taken. All conditions in the `from` statement must match for the action to be taken. The order in which you specify match conditions is not important, because a packet must match all conditions in a term.

If you specify no match conditions in a term, that term matches all packets.

An individual condition in a `from` statement can contain a list of values. For example, you can specify numeric ranges or multiple source or destination addresses. When a condition defines a list of values, a match occurs if one of the values in the list matches the packet.

Individual conditions in a `from` statement can be negated. When you negate a condition, you are defining an explicit mismatch. If a packet matches a negated condition, it is immediately considered not to match the `from` statement, and the next term in the filter is evaluated, if there is one; if there are no more terms, the packet is discarded.

Match conditions are grouped into the following categories depending on how you specify the condition:

- Numeric range
- Address filter
- Multiple match conditions
- Bit-field filter

Numeric range filter conditions match packet fields that can be identified by a numeric value, such as port and protocol numbers. For numeric range filter match conditions, you specify a keyword that identifies the condition and a single value or a range of values that a field in a packet must match. Table 8.18 describes the numeric range filter match conditions. You can specify the numeric range value in one of the following ways:

- Single number (for example, `source-port 25`)
- Range of numbers (for example, `source-port 1024-65535`)
- Text synonym for a single number (for example, `source-port smtp`)

Table 8.18 *Numeric Range Firewall Filter Match Conditions*

Match Condition	Description
keyword-except	Negate a match. For example, destination-port-except *number*.
destination-port *number*	TCP or UDP destination port field. You cannot specify both the port and destination-port match conditions in the same term. Normally, you specify this match in conjunction with the protocol match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify one of the following text synonyms (the port numbers are also listed): afs (1483), bgp (179), biff (512), bootpc (68), bootps (67), cmd (514), cvspserver (2401), dhcp (67), domain (53), eklogin (2105), ekshell (2106), exec (512), finger (79), ftp (21), ftp-data (20), http (80), https (443), ident (113), imap (143), kerberos-sec (88), klogin (543), kpasswd (761), krb-prop (754), krbupdate (760), kshell (544), ldap (389), login (513), mobileip-agent (434), mobilip-mn (435), msdp (639), netbios-dgm (138), netbios-ns (137), netbios-ssn (139), nfsd (2049), nntp (119), ntalk (518), ntp (123), pop3 (110), pptp (1723), printer (515), radacct (1813), radius (1812), rip (520), rkinit (2108), smtp (25), snmp (161), snmptrap (162), snpp (444), socks (1080), ssh (22), sunrpc (111), syslog (514), tacacs-ds (65), talk (517), telnet (23), tftp (69), timed (525), who (513), xdmcp (177), zephyr-clt (2103), or zephyr-hm (2104).
dscp *number*	Differentiated Services code point (DSCP). The Diffserv protocol uses the ToS byte in the IP header. The most significant six bits of this byte form the DSCP. For more information, see Chapter 6, "Interfaces and Class of Service," on page 185. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): The Expedited Forwarding RFC defines one code point: ef (46). The Assured Forwarding RFC defines four classes, with three drop precedences in each class, for a total of 12 code points: af11 (10), af12 (12), af13 (14); af21 (18), af22 (20), af23 (22); af31 (26), af32 (28), af33 (30); af41 (34), af42 (36), af43 (38).
fragment-offset *number*	Fragment offset field.

Table 8.18 *Numeric Range Firewall Filter Match Conditions*

Match Condition	Description
`icmp-code` *number*	ICMP code field. This value or keyword provides more specific information than the `icmp-type`. Because the value's meaning depends on the associated `icmp-type`, you must specify the `icmp-type` along with the `icmp-code`. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed). The keywords are grouped by the ICMP type with which they are associated: parameter-problem: `ip-header-bad` (0), `required-option-missing` (1); redirect: `redirect-for-host` (1), `redirect-for-network` (0), `redirect-for-tos-and-host` (3), `redirect-for-tos-and-net` (2); time-exceeded: `ttl-eq-zero-during-reassembly` (1), `ttl-eq-zero-during-transit` (0); unreachable: `communication-prohibited-by-filtering` (13), `destination-host-prohibited` (10), `destination-host-unknown` (7), `destination-network-prohibited` (9), `destination-network-unknown` (6), `fragmentation-needed` (4), `host-precedence-violation` (14), `host-unreachable` (1), `host-unreachable-for-TOS` (12), `network-unreachable` (0), `network-unreachable-for-TOS` (11), `port-unreachable` (3), `precedence-cutoff-in-effect` (15), `protocol-unreachable` (2), `source-host-isolated` (8), `source-route-failed` (5).
`icmp-type` *number*	ICMP packet type field. Normally, you specify this match in conjunction with the `protocol` match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `echo-reply` (0), `echo-request` (8), `info-reply` (16), `info-request` (15), `mask-request` (17), `mask-reply` (18), `parameter-problem` (12), `redirect` (5), `router-advertisement` (9), `router-solicit` (10), `source-quench` (4), `time-exceeded` (11), `timestamp` (13), `timestamp-reply` (14), or `unreachable` (3).
`interface-group` *group-number*	Interface group on which the packet was received. An interface group is a set of one or more logical interfaces. For information, see "Applying Firewall Filters to Interfaces," on page 361.
`packet-length` *bytes*	Length of the received packet, in bytes. The length refers only to the IP packet, including the packet header, and does not include any Layer 2 encapsulation overhead.
`port` *number*	TCP or UDP source or destination port field. You cannot specify both the `port` match and either the `destination-port` or `source-port` match conditions in the same term. Normally, you specify this match in conjunction with the `protocol` match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify one of the text synonyms listed under `destination-port`.
`precedence` *ip-precedence-field*	IP precedence field. In place of the numeric field value, you can specify one of the following text synonyms (the field values are also listed): `critical-ecp` (0xa0), `flash` (0x60), `flash-override` (0x80), `immediate` (0x40), `internet-control` (0xc0), `net-control` (0xe0), `priority` (0x20), or `routine` (0x00).

Table 8.18 *Numeric Range Firewall Filter Match Conditions*

Match Condition	Description
`protocol` *number*	IP protocol field. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `egp` (8), `esp` (50), `gre` (47), `icmp` (1), `igmp` (2), `ipip` (4), `ipv6` (41), `ospf` (89), `pim` (103), `rsvp` (46), `tcp` (6), or `udp` (17).
`source-port` *number*	TCP or UDP source port field. You cannot specify the `port` and `source-port` match conditions in the same term. Normally, you specify this match in conjunction with the `protocol` match statement to determine which protocol is being used on the port. In place of the numeric field, you can specify one of the text synonyms listed under `destination-port`.

To specify multiple values in a single match condition, group the values within square brackets following the keyword (for example, `source-port [smtp ftp-data 25 1024-65535])`.

To exclude a numeric value, append the string `-except` to the match keyword.

Address filter conditions match prefix values in a packet, such as IP source and destination prefixes. For address filter match conditions, you specify a keyword that identifies the field and one or more prefixes of that type that a packet must match. Table 8.19 describes the address filter match conditions. You can specify the address as a single prefix (a match occurs if the value of the field matches the prefix) or as multiple prefixes (a match occurs if any one of the prefixes in the list matches the packet). To specify the address prefix, use the notation *`prefix/prefix-length`*. To exclude a prefix, specify the string `except` after the prefix. Because the prefixes are order-independent and use longest-match rules, shorter prefixes subsume longer ones as long as they are the same type (whether you specify `except` or not). This is because anything that would match the longer prefix would also match the shorter one.

Table 8.19 *Address Firewall Filter Match Conditions*

Match Condition	Description
`address` *prefix*	IP source or destination address field
`destination-address` *prefix*	IP destination address field
`destination-prefix-list` *prefix-list*	IP destination prefix list field

Table 8.19 *Address Firewall Filter Match Conditions*

Match Condition	Description
`prefix-list` *`prefix-list`*	IP source or destination prefix list field
`source-address` *`prefix`*	IP source address field
`source-prefix-list` *`prefix-list`*	IP source prefix list field

Bit-field filter conditions match packet fields if particular bits in those fields are or are not set. You can match the IP options, TCP flags, and IP fragmentation fields. For bit-field filter match conditions, you specify a keyword that identifies the field and tests to determine that the option is present in the field. Table 8.20 describes the bit-field match conditions. To specify the bit-field value to match, enclose the value in quotation marks (double quotes). Generally, you specify the bits being tested using keywords. Bit-field match keywords always map to a single bit value. You also can specify bit fields as hexadecimal or decimal numbers. To negate a match, precede the value with an exclamation point. To match multiple bit-field values, use the logical operators list in Table 8.21. The operators are listed in order, from highest precedence to lowest precedence. Operations are left-associative.When you specify a numeric value that has more than one bit set, the value is treated as a logical AND of the set bits. You can use text synonyms to specify some common bit-field matches. You specify these matches as a single keyword.

If you specify a port match condition or a match of the ICMP type or TCP flags field, there is no implied protocol match. If you use one of the following match conditions in a term, you should explicitly specify the protocol in the same term:

- `destination-port`—Specify the `protocol tcp` or `protocol udp` match condition in the same term.
- `icmp-code`—Specify the `protocol icmp` match condition in the same term.
- `icmp-type`—Specify the `protocol icmp` match condition in the same term.
- `port`—Specify the `protocol tcp` or `protocol udp` match condition in the same term.

Table 8.20 *Bit-Field Firewall Filter Match Conditions*

Match Condition	Description
Conditions with Variables	
`fragment-flags` *number*	IP fragmentation flags. In place of the numeric field value, you can specify one of the following keywords (the field values are also listed): `dont-fragment` (0x4000), `more-fragments` (0x2000), or `reserved` (0x8000).
`ip-options` *number*	IP options. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `loose-source-route` (131), `record-route` (7), `router-alert` (148), `strict-source-route` (137), or `timestamp` (68).
`tcp-flags` *number*	TCP flags. Normally, you specify this match in conjunction with the `protocol` match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `ack` (0x10), `fin` (0x01), `push` (0x08), `rst` (0x04), `syn` (0x02), or `urgent` (0x20).
Text Synonyms	
`first-fragment`	First fragment of a fragmented packet. This condition does not match unfragmented packets.
`is-fragment`	Matches if the packet is a fragment.
`tcp-established`	TCP packets other than the first packet of a connection. This is a synonym for "`(ack | rst)`". This condition does not implicitly check that the protocol is TCP. To check this, specify the `protocol tcp` match condition.
`tcp-initial`	First TCP packet of a connection. This is a synonym for "`(syn & !ack)`". This condition does not implicitly check that the protocol is TCP. To check this, specify the `protocol tcp` match condition.

Table 8.21 *Bit-Field Logical Operators*

Logical Operator	Description
`(...)`	Grouping
`!`	Negation
`&` or `+`	Logical AND
`|` or `,`	Logical OR

- `source-port`—Specify the `protocol tcp` or `protocol udp` match condition in the same term.
- `tcp-flags`—Specify the `protocol tcp` match condition in the same term.

When examining match conditions, the policy framework software tests only the specified field itself. It does not also test the IP header to determine that the packet is indeed an IP packet. If you do not explicitly specify the protocol when using the fields listed above, design your filters carefully to ensure that they are performing the expected matches.

Applying Firewall Filters to Interfaces

For a firewall filter to work, you must apply it to at least one interface:

```
[edit interfaces]
interfaces interface-name {
  unit logical-unit-number {
    family inet {
      filter {
        input filter-name;
        output filter-name;
      }
    }
  }
}
```

In the `input` statement, list the name of one firewall filter to be evaluated when packets are received on the interface. In the `output` statement, list the name of one firewall filter to be evaluated when packets are transmitted on the interface. You can apply only one input and one output firewall filter to each interface. You can use the same filter one or more times. Input or output filters applied to the loopback interface, `lo0`, affect only input or outbound traffic sent from the Routing Engine, respectively.

When you apply a firewall filter to multiple interfaces, you can name individual counters specific to each interface. These counters enable you to easily maintain statistics on the traffic transiting the different interfaces. Configuration of interface-specific counters also creates

separate instances of any policers you have configured for the same interface. To configure interface-specific counters, include the `interface-specific` statement:

```
[edit firewall filter filter-name]
interface-specific;
```

When applying a firewall filter, you can define an interface to be part of an *interface group*. Packets received on that interface are tagged as being part of the group. You then can match these packets using the `interface-group` match statement. To define the interface to be part of an interface group, include the `group` statement:

```
[edit interfaces interface-name unit logical-unit-number
family inet filter]
group group-number;
```

Configuring Policing

Policing, or rate limiting, enables you to limit the amount of traffic that passes into or out of an interface. It is an essential component of filters designed to thwart DoS attacks. Policing applies two types of rate limits on the traffic: bandwidth, which is the number of bits per second permitted, on average, and maximum burst size. Policing uses a *token-bucket algorithm*, which enforces a limit on average bandwidth while allowing bursts up to a specified maximum value. It offers more flexibility than a *leaky bucket algorithm* in allowing a certain amount of bursty traffic before it starts discarding packets.

You define specific classes of traffic on an interface, to which you can apply a set of rate limits. To do this, you define *policers* within a filter statement. For example, to limit all ftp traffic from a particular source to certain rate limits configure the following:

1. Include one or more policer statements in the filter configuration; they must precede the term definitions. To avoid the time-consuming process of configuring a policer within each filter, you can also define a policer outside the filter; the policer can then be used as a template.
2. Reference the policers in the `then` clause of a term.
3. Add actions, such as accept or discard, or other action modifiers, such as count or log.

4. Apply the policers to an interface for them to be activated. You apply policers the same way you apply firewall filters.

The policer is applied to the packet first, and if the packet exceeds the defined limits, the actions of the `then` clause of the policer are applied. If the result of the policing action was not a discard, the remaining components of the `then` clause of the term are applied.

To specify the rate-limiting part of a policer, include an `if-exceeding` statement, specifying the bandwidth limit in bits per second and the burst size limit in bytes:

```
if-exceeding {
  bandwidth-limit rate;
  burst-size-limit bytes;
}
```

There is no absolute minimum value for the bandwidth limit, and the maximum value is 4.29 Gbps. Any value below 61,040 bps results in a minimum effective rate of 30,520 bps. The maximum value for the burst size limit is 100 MB. The preferred method for setting this limit is to multiply the bandwidth of the interface on which you are applying the filter by the amount of time you allow a burst of traffic at that bandwidth to occur: for example, 5 milliseconds. If you do not know the interface bandwidth, you can multiply the MTU of the traffic on the interface by 10 to obtain a value. If a packet does not exceed its rate limits, it is processed further without being affected. If the packet exceeds its limits, it can be discarded or marked for subsequent processing as specified in the `loss-priority` and `forwarding-class` statements.

To configure a policer action, include the following statements:

```
policer policer-name {
  then {
    policer-action;
  }
```

To simply discard a packet that exceeds the rate limits:

```
then {
  discard;
}
```

To set the loss priority equal to low:

```
then {
  loss-priority low;
}
```

To set the forwarding class:

```
then {
  forwarding-class class-name;
}
```

The possible values for `loss-priority` are `any`, `low`, and `high`, and *`class-name`* is any class name already configured for the forwarding class.

Configure Accounting

For more information, see the JUNOS *Getting Started* technical documentation.

Juniper Networks routers can collect various kinds of data about traffic passing through the router. You can set up one or more *accounting profiles* that specify some common characteristics of this data, including the fields used in the accounting records, the number of files that the router retains before discarding, the number of bytes per file, and the polling period that the system uses to record the data. You configure the profiles using statements at the `[edit accounting-options]` hierarchy level. You assign a unique accounting-profile name for each profile, and this name cross-references the information specified at the `[edit accounting-options]` hierarchy with `interfaces` or `firewall` configuration statements.

Configuring Filter-Based Forwarding

You can configure filters to classify packets based on source address and specify the forwarding path the packets take within the router. You can use this filter for applications to differentiate traffic from two clients that share a common access layer (for example, a Layer 2 switch) but are connected to different ISPs. When the filter is applied, the router can differentiate the two traffic streams and direct each to the appropriate network. Depending on the client's media type, the filter can use the source IP address to forward the traffic to the corresponding network through a tunnel. You can also configure filters to classify packets based on IP protocol type or IP precedence bits. You can forward packets based on input filters only; you cannot forward packets based on output filters. To direct traffic meeting defined match conditions to a specific routing table, include the `routing-instance` statement:

```
[edit firewall filter filter-name term term-name then]
routing-instance routing-instance;
```

See Chapter 9, "Routing and Routing Protocols," on page 373.

To implement filter-based forwarding, you must create a routing table group that adds interface routes to the routing instance created to direct traffic that meets defined match conditions to a specific routing table and to the default routing table `inet.0`. You create a routing table group to resolve the routes installed in the routing instance to directly connected next hops on that interface.

Configuring Traffic Sampling and Forwarding

On routers with an Internet Processor II ASIC, you can sample IP traffic based on particular input interfaces and various fields in the packet header. You can use traffic sampling to monitor any combination of specific logical interfaces, specific protocols on one or more interfaces, a range of addresses on a logical interface, or individual IP addresses. Information about the sampled packets is saved to files on the router's hard disk. The traffic sampling feature is not meant to capture all packets received by a router. Juniper Networks does not recommend excessive sampling (a rate greater than 1 in 1,000 packets), because it can increase the load on the processor. If you need to set a higher sampling rate to diagnose a particular problem or type of traffic received, we recommend that you revert to a lower sampling rate after the problem or troublesome traffic is discovered.

To configure traffic sampling, perform at least the following tasks:

1. Create a firewall filter:

```
[edit firewall]
filter filter-name {
  term term-name {
    then {
      sample;
      accept;
    }
  }
}
```

2. Apply the filter to the logical interfaces on which you want to sample traffic:

```
[edit interfaces]
interface-name {
  unit logical-unit-number {
    family inet {
      filter {
        input filter-name;
      }
      address address {
        destination destination-address;
      }
    }
  }
}
```

3. Enable sampling, specifying a nonzero sampling rate:

```
[edit forwarding-options]
sampling {
  input {
    family inet inet {
      rate number;
    }
  }
}
```

To configure other forwarding options, include one or more of the following statements:

```
[edit forwarding-options]
hash-key {
  family inet {
    layer-3;
    layer-4;
  }
  family mpls {
    label-1;
    label-2;
  }
}
sampling {
  disable;
  input {
    family inet {
      max-packets-per-second number;
      rate number;
      run-length number;
    }
  }
```

```
output {
  cflowd host-name {
    aggregation {
      autonomous-system;
      destination-prefix;
      protocol-port;
      source-destination-prefix {
        caida-compliant;
      }
      source-prefix;
    }
    autonomous-system-type (origin | peer);
    (local-dump | no-local-dump);
    port port-number;
    version format;
  }
  file {
    filename filename;
    files number;
    size bytes;
    (stamp | no-stamp);
    (world-readable | no-world-readable);
  }
  port-mirroring {
    interface interface-name;
    next-hop address;
  }
}
traceoptions {
  file filename {
    files number;
    size bytes;
    (world-readable | no-world-readable);
  }
}
}
```

Configuring Per-Flow Load Balancing Information

You can specify what information the router uses for per-flow load balancing based on port data rather than based only on source and destination IP addresses. For aggregated Ethernet and aggregated SONET interfaces, you can load balance based on the MPLS label information. By default, the software ignores port data when deter-

mining flows. To enable per-flow load balancing, set the `load-balance per-packet` action in the routing policy configuration. To include port data in the flow determination, include the `family inet` statement:

```
[edit forwarding-options hash-key]
family inet {
  layer-3;
  layer-4;
}
```

By default, the router uses the following Layer 3 information in the packet header to load-balance: source IP address, destination IP address, and protocol. If you include both the `layer-3` and `layer-4` statements, the router uses the source IP address, destination IP address, protocol, source port number, destination port number, and incoming interface index to load balance. This is appropriate behavior for Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) packets. For ICMP packets, the field location offset is the checksum field, which makes each ping packet a separate "flow." This can be problematic; for example, some `traceroute` implementations might use ICMP rather than UDP for the outgoing packets.

Configuring Traffic Sampling Output Files

To collect sampled packets in a file in the `/var/tmp` directory, include the `file` statement. Traffic sampling output is saved to an ASCII text file, with each line containing information for one sampled packet.

```
[edit forwarding-options sampling output]
  file {
    filename filename;
    files number;
    size bytes;
    (stamp | no-stamp);
    (world-readable | no-world-readable);
  }
}
```

Tracing Traffic Sampling Operations

Tracing operations track all traffic sampling operations and record them in a log file in the `/var/log` directory. By default, this file is named `/var/log/sampled`. The default file size is 128 KB, and 10 files are created before the first one gets overwritten. To trace traffic sampling operations, include the `file` statement:

```
[edit forwarding-options sampling traceoptions]
file filename {
  files number;
  size bytes;
  (world-readable | no-world-readable);
}
```

Configuring Flow Aggregation (cflowd)

You can collect an aggregate of sampled flows and send the aggregate to a specified host that runs the cflowd application available from CAIDA (http://www.caida.org). Using cflowd, you can obtain various types of byte and packet counts of flows through a router. The cflowd application collects the sampled flows over a period of 1 minute. At the end of the minute, the number of samples to be exported are divided over the period of another minute and are exported over the course of the same minute. By default, flow aggregation is disabled. To enable the collection of flow aggregates, include the `cflowd` statement, specifying the name or identifier of the host that collects the flow aggregates.

```
[edit forwarding-options sampling output]
cflowd host-name {
  aggregation {
    autonomous-system;
    destination-prefix;
    protocol-port;
    source-destination-prefix {
      caida-compliant;
    }
    source-prefix;
  }
  autonomous-system-type (origin | peer);
  (local-dump | no-local-dump);
  port port-number;
  version format;
}
```

You must also include the UDP port number on the host and the `version`, which gives the format of the exported cflowd aggregates. To collect cflowd records in a log file before exporting, include the `local-dump` statement. To specify aggregation of specific types of traffic, which conserves memory and bandwidth in enabling cflowd to export targeted flows rather than all the aggregated traffic, include the `aggregation` statement. The aggregation type can be one of the following:

- `autonomous-system`—Aggregate by AS number.
- `destination-prefix`—Aggregate by destination prefix only.
- `protocol-port`—Aggregate by protocol and port number; requires setting the separate `cflowd port` statement.
- `source-destination-prefix`—Aggregate by source and destination prefix.
- `source-prefix`—Aggregate by source prefix only.

Collection of sampled packets in a local ASCII file is not affected by the `cflowd` statement.

To collect the cflowd flows in a log file before they are exported, include the `local-dump` statement. By default, the flows are collected in `/var/log/sampled`. Note that you cannot configure both host (cflowd) sampling and port mirroring at the same time.

Configuring Port Mirroring

On routers containing an Internet Processor II ASIC, you can send a copy of an IPv4 packet from the router to an external host address or a packet analyzer for analysis, also known as *port mirroring*. Port mirroring is different from traffic sampling. In traffic sampling, a sampling key based on the IPv4 header is sent to the Routing Engine, and the key can be placed in a file or cflowd packets based on the key can be sent to a cflowd server. In port mirroring, the entire packet is copied and sent out through a next-hop interface. To configure port mirroring, configure traffic sampling on a logical interface by including the `input` statement at the `[edit forwarding-options sampling]` hierarchy level. Then specify the output interface to the analyzer and port-mirroring destination in the `port-mirroring` statement:

```
[edit forwarding-options sampling output]
port-mirroring {
  interface interface-name;
  next-hop address;
}
```

The following restrictions apply to port mirroring:

- You cannot configure both cflowd sampling and port mirroring in the same configuration.
- You cannot configure firewall filters on the port-mirroring interface.
- The interface you configure for port mirroring should not participate in any kind of routing activity.
- The destination address should not have a route to the ultimate traffic destination. For example, if the sampled IPv4 packets have a destination address of `190.68.9.10` and the port-mirrored traffic is sent to `190.68.20.15` for analysis, the device associated with the latter address should not know a route to `190.68.9.10`. Also, it should not send the sampled packets back to the source address.
- Only IPv4 traffic is supported.
- You can configure only one port-mirroring interface per router. If you include more than one interface in the `port-mirroring` statement, the previous one is overwritten.
- You must include a firewall filter with both the `accept` action and the `sample` action modifier on the inbound interface for port mirroring to work. Do not include the `discard` action or port mirroring does not work.