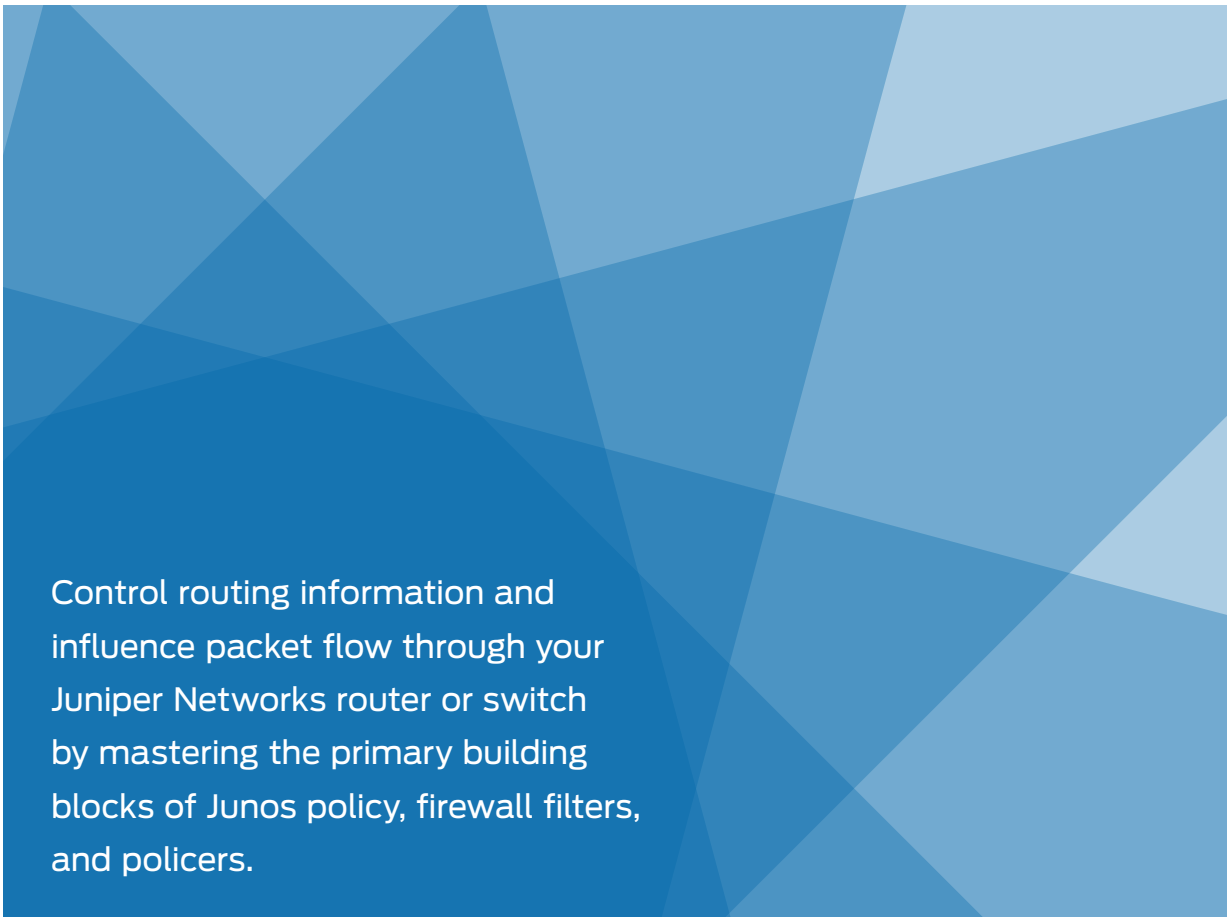
JUNIPER NETWORKS

Junos® Fundamentals Series

DAY ONE: CONFIGURING JUNOS POLICY AND FIREWALL FILTERS



By Jack W. Parks, IV

DAY ONE: CONFIGURING JUNOS POLICY AND FIREWALLL FILTERS

Pairing routing policy and firewall filters may, at first glance, seem like an odd combination for a routing book. After all, filters are for security and policy is about manipulating route attributes and readvertisement.

While route advertisement decisions can impact security, these two topics are more logically bundled into a single book because of the high degree of similarity in their Junos configuration syntax. Knowing one simply helps you learn the other, and given that both are critically important topics in modern IP networks, their synergy should not be ignored.

Day One: Configuring Junos Policies and Firewall Filters shows how the savvy network administrator can make unified and robust efficiencies using two similar tools from their Junos toobox.

> "Jack Parks provides clear, concise descriptions and configuration examples to illustrate basic concepts as well as complex examples that demystify policy and filter operations and capabilities that are not widely understood. This is your chance to finally understand why that nested firewall or Boolean grouped policy did not behave as you expected."
>
> *Harry Reynolds, Author, Senior Test Engineer, Juniper Networks*

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Describe the features of policy, firewall filters, and policers in Junos.
- Understand the differences between policy and firewall filters.
- Configure policy, firewall filters, and policers in the Junos CLI.
- Create useful policies for your network.
- Understand how policy flow and default policy actions work in Junos.
- Develop a foundation for advanced routing policy topics.
- Create hierarchical policy and chain policy together.
- Create routing policies that share or filter routes with other routers in the network.
- Understand the configuration as it relates to firewall filters and policers and the benefits of using them in your network.

Juniper Networks Books are singularly focused on network productivity and efficiency. Peruse the complete library at www.juniper.net/books.

Published by Juniper Networks Books

ISBN 978-1936779369

51600

9 781936 779369

07100143

JUNIPER NETWORKS

Junos® Fundamentals Series

Day One: Configuring Junos Policy and Firewall Filters

By Jack W. Parks, IV

JUNIPER NETWORKS

Published by Juniper Networks Books
Author: Jack W. Parks
Technical Reviewers: Peter Van Oene
Editor in Chief: Patrick Ames
Copyeditor and Proofer: Nancy Koerbel
J-Net Community Manager: Julie Wider

About the Author
Jack W. Parks, IV has worked since 1992 in almost every position known in the realm of IT. After serving eight years in the United States Air Force, Jack transitioned to the corporate world and worked in the large Enterprise and ISP market spaces. Most recently he has focused on Enterprise Routing and Switching, Service Provider Routing, MPLS, and VPNs. With a B.S. in Business Information Systems from John Brown University and several industry certifications, including CCIE #11685 & JNCIE-M #666, Jack is currently a Juniper Networks Systems Engineer based in Atlanta, Georgia.

Author's Acknowledgments
Many thanks to my technical editor and mentor Peter Van Oene. Thanks to the *Day One* team for the opportunity and encouragement to develop another book. And to my family: thank you for giving up your nights and weekends with me so I could finish this project.

ISBN: 978-1-936779-38-3 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-39-0 (ebook)

Version History: v1 September 2011
2 3 4 5 6 7 8 9 10 #7100143-en

This book is available in a variety of formats at: www.juniper.net/dayone.

Send your suggestions, comments, and critiques by email to dayone@juniper.net.

What You Need to Know Before Reading this Book

- You should have basic knowledge of Junos CLI, its syntax, and its hierarchy. It is recommended that you have read the *Junos Fundamental DayOne Series*.
- You should have an understanding of basic packet filtering principles and policing fundamentals.
- You should understand the difference between route (prefix) filtering and packet filtering.

After Reading this Book, You'll be Able To...

- Describe the features of policy, firewall filters, and policers in Junos.
- Understand the differences between policy and firewall filters.
- Configure policy, firewall filters, and policers in the Junos CLI.
- Create useful policies for your network.
- Understand how policy flow and default policy actions work in Junos.
- Develop a foundation for advanced routing policy topics.
- Create hierarchical policy and chain policy together.
- Create routing policies that share or filter routes with other routers in the network.
- Understand the configuration as it relates to firewall filters and policers and the benefits of using them in your network.

The *Day One* Book Series

This book is part of a growing library of *Day One* books, produced and published by Juniper Networks Books.

Day One books were conceived to help you get just the information that you need on day one. The series covers Junos OS and Juniper Networks networking essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

The *Day One* library also includes a slightly larger and longer suite of *This Week* books, whose concepts and test bed examples are more similar to a weeklong seminar.

You can obtain either series, in multiple formats:

- Download a free PDF edition at http://www.juniper.net/dayone.
- Get the ebook edition for iPhones and iPads from the iTunes Store. Search for *Juniper Networks Books*.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Kindle Store. Search for *Juniper Networks Books*.
- Purchase the paper edition at either Vervante Corporation (www.vervante.com) or Amazon (www.amazon.com) for between $12-$28, depending on page length.
- Note that Nook, iPad, and various Android apps can also view PDF files.
- If your device or ebook app uses .epub files, but isn't an Apple product, open iTunes and download the .epub file from the iTunes Store. You can now drag and drop the file out of iTunes onto your desktop and sync with your .epub device.

Chapter 1

Policy and Firewall Filters Introduction

An often-heard grumble is that Juniper Networks applies new and strange definitions to existing networking concepts when discussing the Junos operating system and its features, two of which are the topic of this book: policy and firewall filters. The thing is, how Junos interprets these terms is closely related to the actual industry terminology found in documents like RFCs and BCPs, but in your travels with other operating systems or other networking equipment, you may have strayed from the open standards that Junos so closely follows.

TIP Don't worry if you encounter unfamiliar usages and even terminology as you work through this book – the concepts you need to understand about Junos policy and firewall filters are provided by using solid examples all along the way, so you can see the concepts in action and not just be told about them.

Let's begin with how the Junos OS defines *policy* and *firewall filters* and how it uses them.

What is Policy?

Policy is used to control the flow of routing information between routing processes and the routing table. Policy is also used to add, remove, or modify attributes associated with the routing information, thereby controlling the size and scope of the routing information available to a networked device.

In simple terms, policy is what allows static routes to be advertised by OSPF to its neighbors, or BGP to prepend AS-PATH information to its peer routers. Any time routing information needs to be shared between protocols, policy is employed. Filtering information between neighbors is another function of policy. If there is a requirement to manipulate the flow of routing information, then policy is the tool to accomplish that task.

What are Firewall Filters?

Firewall filters are stateless filtering policies used to control the flow of individual packets. A popular use for firewall filters is to filter, or drop, packets from the transit data stream.

TIP Still confused about what a firewall filter is? Maybe it would be helpful if you referred to it by a common industry name – *access control list* (ACL).

Don't be confused by the word "firewall" here. Traditionally you might think of a firewall as being a specialized networking appliance that keeps track of flows and blocks unwanted traffic from entering the network. This assumes that a firewall is stateful, but there are many types of firewalls and the Junos firewall filter is a *stateless* packet filter, and it is not limited to just discarding packets. Packet classification, counting, sampling, rate limiting, and logging are other capabilities of a Junos firewall filter.

Quick Comparison of Policy and Firewall Filters

So policies and firewall filters are very similar in syntax, even though they have different purposes in Junos operation. Policy is used to control routing information, which *indirectly* influences packet flow through the router or switch. Firewall filters affect packet flow *directly* by taking action on individual packets as they traverse the router or switch.

NOTE In Junos, firewall filters are technically policies, which is why they are presented concurrently in this book as well as in Juniper Networks Technical Documentation. This book, however, tries to avoid mentioning the word "policy" when discussing firewall filters in order to minimize confusion.

Even though policy and firewall filters are contained under different configuration stanzas in Junos, the configuration architecture is the same. It's the purpose and implementation differences that separate them. The primary building block of both policy and firewall filters is the "term." Functions are grouped into terms and it is those terms that are evaluated, in sequential order, to determine the outcome of the policy. Terms contain the match conditions as well as the associated actions if the match conditions are met.

MORE? If you need a more comprehensive comparison of policy and firewall filters, then check out *Comparison of Routing Policies and Firewall Filters*, at http://www.juniper.net/techpubs/en_US/junos10.4/topics/reference/general/policy-routing-policies-firewall-filters-comparison.html.

Syntax and Flow of Policy

The beautiful thing about Junos policy is how it can be defined from a requirement expressed in written or spoken English. It follows the logical progression of "if" this condition is true "then" take the following actions.

For example, a simple statement such as "the IP prefix 10.10/16 should have a metric of 10" can be used to produce the following policy configuration:

```
[edit policy-options policy-statement some-test-policy]
jack# show
term plain-english {
   from {
      route-filter 10.10.0.0/16 exact;
   }
   then {
      metric 10;
      accept;
   }
}
```

It is also possible to translate the desired function of the policy back into English from the same configuration. This policy reads that if a prefix matches 10.10/16 then it would set the metric to 10 and accept the prefix. This is a simple example and a small introduction to policy so let's explore the syntax in a little more depth.

TIP Policies are not specific to any particular routing protocol, like BGP or OSPF. A well-constructed policy may be applied to multiple protocols simultaneously.

There are two steps to using policy in Junos. The first step is defining what the policy must do, and the preceding example is an illustration of such defining. The second step is applying the policy to a routing protocol to call the policy into action. Understanding how the policy works when applied to routing protocols is crucial to avoid unintended consequences – like network disruptions.

TIP Unintended consequences, also known as side effects, are common when first learning Junos policy. Testing and troubleshooting tools are discussed later in this book, but for now, know that policy should be fully vetted before it is used in the network. If you are following along with this book on a device, use a lab or testbed.

Policy Syntax Components

Policy Name

The policy name is the topmost container in the Junos syntax and identifies the entire policy. It's what's referenced when applying the policy to routing protocols. Policy names are *user-defined* variables in Junos, and picking a meaningful and descriptive name really helps identify the policy's purpose and application.

Policy names are defined under the `[policy-options]` hierarchy as a `policy-statement`:

```
[edit policy-options]
jack# edit policy-statement some-test-policy
```

TIP In Junos, user-defined variables may be auto completed in the CLI by pressing the Tab key. If you are used to the IOS CLI, you might be inclined to use naming conventions that are short and to the point, because without tab completion you have to memorize the names in order to type them in by hand when referencing them. The Junos CLI allows tab-completion of user-defined variables in the same way as is done for system variables, thus allowing for more meaningful naming syntax for policies, term, filters, etc.

Term

Terms group any match conditions and actions together under a common hierarchy in the configuration. There are two subsections within a term, the "from" statement and the "then" statement. The "from" statement is used as the match criteria for a term. The action that the term will take is under the "then" statement. It's easy to discern the difference between the two term components, for example:

```
[edit policy-options policy-statement some-test-policy]
jack# show
term plain-english {
    from {
        route-filter 10.10.0.0/16 exact;
    }
    then {
        metric 10;
        accept;
    }
}
```

It is possible, and highly probable, to have more than one match condition per term, but the action statement follows slightly different

rules. There can be only a single terminating action, but the action statement can be used to modify several attributes of a prefix at the same time. For example, a policy term can change the metric of a route to 10, *and* add a BGP community, *and* prepend two AS-PATH objects to a prefix; however, only one terminating action such as accept, *or* reject, *or* next can be applied to a single term.

NOTE You might notice that Junos policy allows modification of attributes without terminating the policy chain. This is a significant advantage over IOS policy, as it allows a Junos router to accomplish what would normally require very many specific route-maps with a much smaller set of modular policies.

Policy Flow

The most important thing to remember about policy is that terms are processed in a top down sequential order. Policy actions fall into two major categories: *flow control actions* and *modifier actions*. Policy terms that have a particular type of flow control action, called a *terminating action*, stop the further processing of the policy for a given prefix. A terminating action would be to accept or reject a given set of match criteria.

In Figure 1.1, the prefix 10.10/16 will match Term A and be accepted, and the prefix 10.10/16 will not be evaluated against Terms B and C. The continued processing for 10.10/16 is stopped, but additional prefixes, presumably learned from the same neighbor, are processed until all of the remaining prefixes have been evaluated against the policy and the appropriate termination actions are applied.

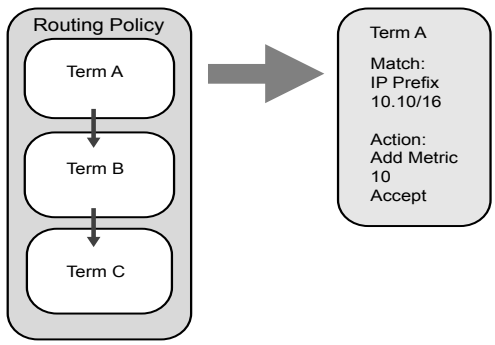


Figure 1.1 Policy Flow

Let's apply Figure 1.1's policy flow in the real world. If a router has ten routes to be evaluated by the above policy then each one will be checked against Term A, Term B, and Term C until a match condition is satisfied and a terminating action is applied or all terms are processed.

Ten routes: 10.10/16, 10.11/16, 10.12/16 … 10.19/16.

Note that 10.10/16 will match Term A and further processing of 10.10/16 will stop with this policy. 10.11/16 will not match Term A so the next term, which is Term B, will be evaluated, and so on.

Syntax and Flow of Firewall Filters

Now firewall filters are very similar to policy in syntax and flow because firewall filters are a type of policy. The main difference between them is that firewall filters act on the packets instead of the routing information.

Let's show an example where you can see the similarity in syntax. The following firewall filter matches GRE packets that fall within the 192.168/16 supernet, then counts the packets, under the name `sample-counter`, and sends the packets to the bit bucket:

```
[edit firewall family inet filter test-firewall-filter]
jack# show
term sample-term {
   from {
      source-address {
         192.168.0.0/16;
      }
      protocol gre;
   }
   then {
      count sample-counter;
      discard;
   }
}
term last-term {
   then accept;
}
```

NOTE A common criticism of Junos firewall filters, as compared to IOS ACLs, is that Junos firewall filters are more involved to configure. But do not mistake the CLI output of a given configuration with the amount of typing required to configure such a feature. The amount of effort required to configure a simple multiline ACL and a Junos firewall filter is almost identical. The readability and search capabilities of the Junos CLI confirm its added value.

In Junos, firewall filters have an explicit default action at the end of all filters, which is to drop all packets. Cisco IOS users, who are experienced with ACLs, understand this default behavior and Junos is no different than IOS ACLs in this regard, which is why the term `last-term` in the above example has an action of `then accept`. It's the equivalent of the IOS ACL line: `accept ip any any`.

Firewall Filter Syntax

Firewall Filter Name

The firewall filter name is the topmost container and identifies the entire access-list. The firewall filter name is what is used to apply the firewall filter to an interface. Filter names are user-defined variables in Junos. Picking a meaningful and descriptive name helps identify the firewall filter's purpose and application.

In Junos, firewall filters are created under the `[firewall]` hierarchy. Best practices dictate that firewall filters should be configured under the appropriate protocol family, which is the family that the ACL will be used for. If the access list will be used to filter IP packets then the filter should be configured under `family inet` as shown here (for IPv6 they should be configured under `family inet6`):

```
[edit firewall]
jack# show
family inet {
   filter test-firewall-filter {
      term sample-term {
         from {
            source-address {
               192.168.0.0/16;
            }
            protocol gre;
         }
         then {
            count sample-counter;
            discard;
         }
      }
      term last-term {
         then accept;
      }
   }
}
```

TIP On some Junos platforms, firewall filters can be configured directly under the `firewall` stanza without placing them under the `[family]` hierarchy.

This syntax only exists for backward compatibility and is not available on newer Junos device platforms. For consistency, it is recommended that you create all firewall filters under the appropriate family class.

Note that in the preceding example, the name of the firewall filter is `test-firewall filter` – far better than something like "100" – and that firewall filter names are configured under `firewall family inet` as a filter.

Term

Terms in a firewall filter are used to group any matching criterion together for a specific action to be applied. Like the policy term, the firewall filter term identifies the matching conditions with the "from" statements, and the actions under the "then" statements:

```
[edit firewall family inet filter test-firewall-filter]
jack# show
term sample-term {
   from {
      source-address {
         192.168.0.0/16;
      }
      protocol gre;
   }
   then {
      count sample-counter;
      discard;
   }
}
term last-term {
   then accept;
}
```

Terms can contain multiple match conditions in Junos, but only a single final action such as accept, reject, or discard. This is a big distinction between Junos and other network operating systems. Single line access lists must repeat match conditions for a given action, while Junos firewall filters allow for the grouping of like match conditions for a common action.

NOTE Advanced firewall options and syntax are covered in Chapter 4.

Firewall Filter Flow

When the Junos OS processes a firewall filter, it does so through a top down process. As shown in Figure 1.2, Term A is processed first, then

Terms B and then Term C. If a match is made in a given term then all processing stops for that filter.

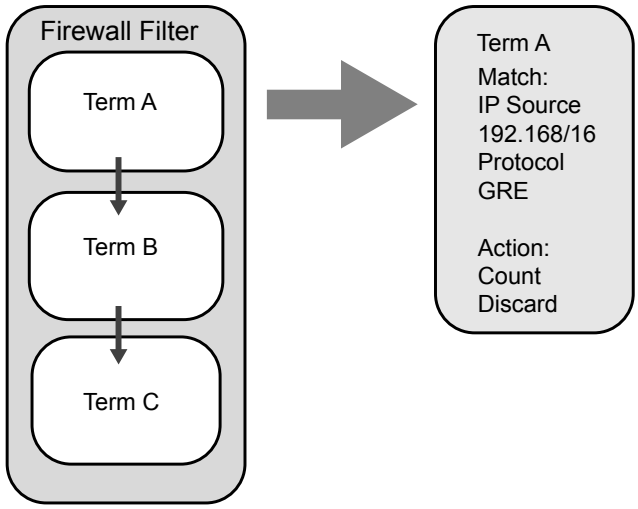


Figure 1.2 Firewall Filter Packet Processing Flow

Figure 1.2 illustrates the packet processing flow of a firewall filter. If a GRE packet with a source IP address of 192.168.1.100 is processed by the firewall filter, then the packet will be counted and dropped. However, if a packet with an IP address of 192.180.2.79 is processed by the filter, then it will not match Term A, but it may match Term B, or Term C. The point being that the firewall filter is evaluated in a top down fashion. How this works is taken up in more detail in Chapter 2.

Summary

This chapter served as a primer on Junos policy and firewall filters and how they differ. The next few chapters dive more deeply into both of these security features in the Junos OS, but first, it's important to set the stage, so to speak.

Policy and firewall filters can become very complex depending on your network and its usage, but not all policy and filters have to be complex, as this book shows.

TIP Follow along with the rest of this book directly connected to a Junos device.

Chapter 2

Policy Configuration

Chapter 1 provided a brief overview of Junos policy, but more discussion is required surrounding the match conditions, actions, and evaluation logic. There are several complimentary components to Junos policy that make it a powerful part of your network operation.

Match Conditions

Match conditions are the "if" statements in policy evaluation. They are not limited to IP prefixes and can be a specific BGP neighbor or an incoming interface.

From versus To

As discussed in Chapter 1, the "from" clause is the actual match condition of the policy. Anything contained under the "from" section represents the match criteria and will be susceptible to the action definitions of the "then" section.

There is another match statement, the "to" clause, which has the same matching effects as the "from" clause. As a rule of thumb, using the "to" clause as match criteria has the same effect as the "from" clause. There are exceptions, however. The most prevalent use of the "to" clause is with IS-IS to affect route leaking between IS-IS levels. Paired with BGP policy, the "to" clause may be used to identify a remote BGP neighbor.

The corner case uses for the "to" clause only have an effect when applied to a protocol that understands neighbor relationships or areas regarding direction. ISIS and BGP both have the ability to use "to" as a match condition, but only with specific match conditions. The use of the "to" clause will act as a "from" when used with conditions that do not have a direction.

Protocol Specific Match Conditions

One of the ways that Junos policy is more vibrant than other network operating systems is the breadth of the match conditions that are available under the "from" clause. Policy is protocol independent, but that doesn't mean that all of the match conditions can be used with any type of route. The following tables break down the match conditions for a given routing protocol.

Table 2.1 Match Conditions for Various Routing Protocols

Protocol Specific Match Conditions: BGP		
Match Condition Name	Extended Values	Extended Parameters
as-path	list contained in brackets	as-path name or regex
local-preference	n/a	n/a
metric	n/a	n/a
preference	n/a	n/a
origin	n/a	n/a

Protocol Specific Match Conditions: ISIS and OSPF (Link State)		
Match Condition Name	Extended Values	Extended Parameters
area (OSPF)	n/a	n/a
external (OSPF)	n/a	n/a
level (ISIS)	n/a	n/a
tag	n/a	n/a

Protocol Specific Match Conditions: RIP		
Match Condition Name	Extended Values	Extended Parameters
interface	list contained in brackets	n/a

Protocol Specific Match Conditions: Non-protocol Dependent		
Match Condition Name	Extended Values	Extended Parameters
as-path (static or aggregate)	list contained in brackets	as-path name or regex
color	n/a	n/a
community	list contained in brackets	community name or regex
interface (direct or local)	n/a	n/a

next-hop	list contained in brackets	n/a
policy	list contained in brackets	n/a
prefix-list	separate entries	n/a
protocol	list contained in brackets	n/a
rib	n/a	n/a
route-filter	separate entries	action
source-address-filter (multicast)	separate entries	action

MORE? For a detailed look at the match conditions not covered in this book, see *Configuring Match Conditions in Routing Policy Terms* at http://www.juniper.net/techpubs/en_US/junos10.4/topics/usage-guidelines/policy-configuring-match-conditions-in-routing-policy-terms.html.

Prefix Lists

Prefix lists contain prefixes, and like most Junos configurations, prefix lists use user-defined names that can be referenced in future configuration stanzas. Prefix lists are a simple and effective configuration device to group prefixes together – typically those prefixes that represent a single purpose.

Prefix lists can be used over and over again in different policies, and in different terms within a particular policy. This can save a lot of typing. For example, the same prefix list can be used in a routing policy for OSPF, BGP, as well as be referenced in a firewall filter. It's nice to have that flexibility.

TIP It is recommended that you group similar IP prefixes together under a single prefix list, like web-servers or exchange-servers. It allows the purpose of the list to be understood at a quick glance. Multiple prefix lists can be applied to same term.

Here's a sample Junos syntax of a prefix list:

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

NOTE A prefix list is called in policy under the "from" statement.

Routes contained within prefix lists are exact match only. If the route 192.168.1.0/24 is configured in a prefix list then only an exact match of 192.168.1.0/24 will be considered. Longer prefixes, such as 192.168.1.4/30 and 192.168.1.128/25, will not be considered a match with respect to the prefix list. And here is an actual example of a prefix list:

```
[edit policy-options]
jack# show
prefix-list typical-pl-example {
   192.168.1.0/24;
   172.16.1.0/24;
}
```

Apply Path

Prefix lists also have a configuration element named *apply path*. Apply path allows you to specify a particular part of the configuration stanza that contains IP addresses to populate the prefix list itself, as shown here:

```
[edit policy-options]
jack# show
prefix-list BGP-Peers-Prefixes {
   apply-path "protocols bgp group <*> neighbor <*>";
}
```

This apply-path example takes all of the neighbors defined under the stanza `[ protocols bgp ]` and adds those host addresses to the prefix list automatically. As you add and delete BGP neighbors, the prefix list is updated automatically. Not only can prefix lists save you repetitive typing, but they can save you from typing, period!

Route Filters

The most common way to match IP prefixes in policy is to use the `route-filter` statement. Route filters are explicitly configured in a policy statement under a term. Unlike a prefix list, route filters are not portable configurations and exist only under the term in which they were configured. If a route filter needs to be used again it must be reconfigured under the new term. That being said, a route filter is more advanced than a prefix list, thanks to the match type for each destination prefix as well as an optional action. The components of a route filter syntax are the following:

```
route-filter destination-prefix match-type <actions>
```

Match Type

What makes the route filter so powerful is that it can match multiple destination prefixes in a single line of code thanks to the *match type section* of its configuration. The simplest of the match conditions is the *exact keyword*, and the most complicated to understand is the *through match type*. There are plenty of match conditions in between. Table 2.2 explains the match types and their functions using the destination prefix 192.168/16 for its examples.

Table 2.2 Match Type Functionality

Match Type Keyword	Keyword Function	Difficulty Rating
exact	Match the destination prefix exactly as defined. If the prefix is defined as 192.168/16 then only the 192.168/16 prefix will satisfy the match condition.	easy
longer	Match every prefix that is longer than the defined prefix but within the supernet. If 192.168/16 is defined and the *longer* match type is configured with it, then all the prefixes that are more specific, excluding the defined 192.168/16 prefix, are considered matches. Prefixes such as 192.168.20/24 and 192.168.128/22 are matches because they are a part of the 192.168/16 supernet. The 192.168/16 prefix is excluded from the match criteria. Think about *longer* as a greater than equation.	moderate
orlonger	The *orlonger* match type is closely related to the match type longer, but it includes the specified destination-prefix. 192.168/16, 192.168.20/24 & 192.168.128/22 are all matches when the orlonger match type is used. Think about *orlonger* as an equal to or greater than.	moderate
prefix-length-range	The match type *prefix-length-range* looks simple on the surface but it can become confusing. The defined destination prefix sets the major supernet for the match condition and the prefix range sets the scope for the match condition. The *prefix-length-range* is configured similar to /22-/24. The prefixes 192.168.20/24 and 192.168.128/22 both match, but 192.168.100.128/29 fall outside of the /22-/24 scope and do not match.	moderate

through	The match type *through* can be complicated to understand. Essentially, through matches the most significant bits of a given destination prefix and the most significant bits of the second destination prefix. *Through* charts a path from one starting prefix and length to a second prefix and length, with a single matching prefix per prefix length.	hard
upto	The match type *upto* works like the orlonger match type but with a predefined stoping point. If a route filter was configured for 192.168/16 upto /24, then all prefixes that fall into the defined range are matches. All prefixes more specific than /24 do not match (for example, /25-/32).	moderate

If a picture is worth a thousand words, then Figure 2.1, below, illustrates each of the match type keywords literally described in Table 2.2. The individual drawings show a radix tree breakdown of the routing table. Each tree begins with the prefix 192.168.0.0/16 at the top of the tree with the applied match type. The highlighted area covers the matched prefixes covered with the match type keyword.

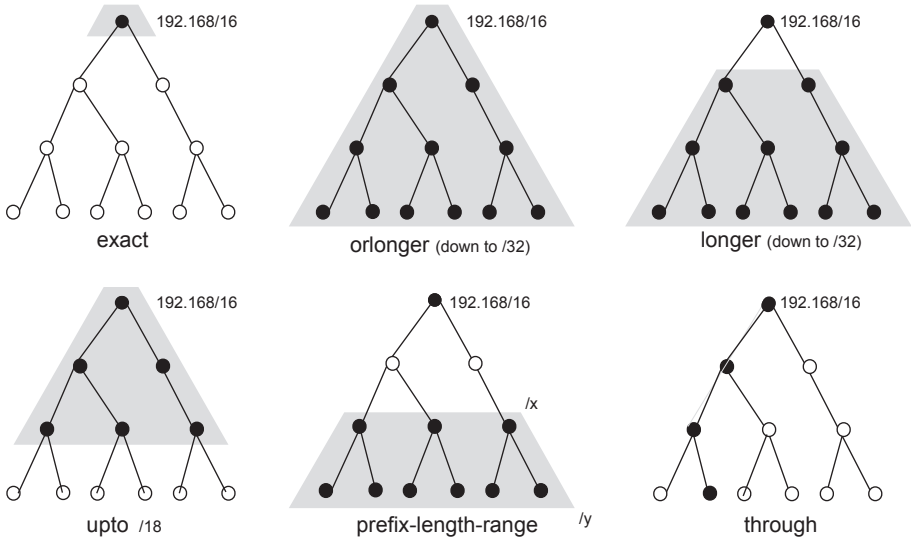


Figure 2.1 Match Type Functionality Illustrated

Optional Actions

Route filters also contain an optional action element. This action component supersedes the term's action statement, or the "then" statement. This may cause unwanted behavior in your policy if used without concern for policy flow. The optional actions available for route-filters include just about any action available to the policy itself. If there are route modifiers under the "then" statement in a policy, such as changing a route metric, and the route filter action is used, then the metric will not be changed.

TIP If you are studying for your advanced Junos certifications, it would be wise to understand how route filter optional actions affect the flow of the policy. You just might be tested on that topic (hint).

Let's drill down on route filter optional actions for a second. Consider this:

```
[edit policy-options]
jack# show
policy-statement some-test-policy {
   term plain-english {
      from {
         route-filter 10.10.0.0/16 exact accept;
      }
      then {
         metric 10;
         accept;
      }
   }
}
```

The route filter optional action used in this policy is `accept`, which terminates the policy flow and further policy actions such as setting the route metric to a value of 10. Beware of, and be aware of, the unintended consequences of using optional statements.

Prefix List Filters

As the name suggests, *prefix list filters* provide a nice hybrid option between the prefix list and the route filter. At the core of the prefix list filter is the prefix list itself, but it uses the route filter-style match type and optional actions. While the match type is not as extensive as with the route filter, it does encompass the most used match types as listed in Table 2.3. The Junos syntax looks like this:

```
prefix-list-filter prefix-list-name match-type <actions>
```

Table 2.3 Prefix List Filter Match Type Functionality

Match Type Keyword	Keyword Function
exact	Match the destination prefix configured within the prefix list exactly as defined.
longer	Match every prefix in the prefix list that is more specific than defined prefix but within the supernet. Think about *longer* as a greater than equation.
orlonger	The *orlonger* match type is closely related to the match type *longer*, but it includes the specified destination-prefix. 192.168/16, 192.168.20/24, and192.168.128/22 are all matches when the *orlonger* match type is used. Think about *orlonger* as an *equal to* or *greater than*.

NOTE Each line in the prefix list is evaluated individually and the match type is evaluated against each prefix respectively.

Boolean Operations

What happens when multiple match conditions are configured under a term? How does Junos determine the order of operations and selection? At its core, policy relies on two Boolean operations: *and* and *or*. A good rule of thumb to differentiate the pair is this: conditions that are presented in a horizontal orientation are regarded as *or*; conditions that are presented vertically are regarded as *and*. Consider the following example:

```
policy-statement boolean-example {
   term boolean-and {
      from {
         protocol bgp;
         neighbor 1.1.1.1;
      }
      then accept;
   }
   term boolean-or {
      from {
         protocol [ bgp ospf direct ];
         prefix-list some-prefixes;
      }
      then accept;
   }
}
```

Here the *and* and *or* operations are displayed by term. Reading the policy and starting with the term *boolean-and*, a match occurs only

when a route matches both the protocol and the neighbor statements. A valid match would have to be learned via BGP from the neighbor 1.1.1.1. The second term, *boolean-or,* matches any routes contained with the prefix-list *some-prefixes*, and is learned from the protocol BGP, or OSPF, or from directly connected interfaces.

TIP Boolean *or* examples are usually contained between two square brackets [ ].

On the other hand, Junos evaluates route filters a bit differently, as a longest match regardless of order. This is important to remember, especially if optional actions are configured. The following code snippet, `boolean-route-filter`, contains three route filter statements. Study it and answer this question: What is the outcome if the route 192.168.24.0/24 is evaluated against this policy?

```
policy-statement boolean-route-filter {
   term boolean-rf {
      from {
         route-filter 172.16.0.0/16 exact;
         route-filter 192.168.0.0/16 exact;
         route-filter 192.168.24.0/24 exact reject;
      }
      then {
         metric 10;
         accept;
      }
   }
}
```

You might quickly say that the route is accepted with a metric of 10, but remember, however, that route filters are evaluated by longest match. The 192.168.24.0/24 prefix would be rejected thanks to the optional action of `reject`.

Advanced Boolean Operations

The Junos OS allows for the use of advanced Boolean operators to assist with policy matching conditions. In general terms, matches occur with single inputs – like from a specific neighbor and a given set of prefixes. What if you could create a deeper logic to determine what constitutes a match and what is rejected? What if you could create compound match conditions? The answers, of course, are the advanced Boolean operators: `&&`, `||` , and `!`.

&& - the *and* operator. It requires at least two of the conditions to be true, or to match.

|| - the *or* operator. This operator ensures that either of the conditions is true.

! – the *not* operator. This operator negates a match.

Advanced Boolean operators are not applied within policy but, rather, the logic is applied to the import and export statements. This pits policy against policy for a given protocol.

To help visualize this concept, let's look at some code and step through the logic process:

```
[edit]
jack# show policy-options
policy-statement match-172-16 {
    term 1 {
        from {
            route-filter 172.16.0.0/16 orlonger;
        }
        then accept;
    }
    term 2 {
        then reject;
    }
}
policy-statement match-172-17 {
    term 1 {
        from {
            route-filter 172.17.0.0/16 orlonger;
        }
        then accept;
    }
    term 2 {
        then reject;
    }
}
[edit]
jack# show protocols
bgp {
    export ( match-172-16 && match-172-17 );
}
```

TIP Don't confuse the operators `()` and `[]`.

There are two policies at play in this example. Each policy only accepts a single prefix range. These policies have been applied to BGP as an export statement utilizing the && Boolean operator.

Understanding the Logic

The advanced Boolean logic works by the policies returning a true or false. A true is returned if a prefix is accepted or encounters the action next-policy. A false is returned if a prefix encounters an action of reject.

Looking at the preceding example policy-statement `match-172-16`, any prefix that matches the 172.16.0.0/16 orlonger will return a value true because the action is accept. Any prefix outside the 172.16.0.0/16 range will be rejected returning the value of false.

TIP The final result of the entire policy that uses Boolean operators must be true too, for the prefix to be accepted.

This first chart shows the logic operator && evaluation.

Policy X	Policy Y	Policy X && Policy Y
true	true	true
true	false	false
false	true	false
false	false	false

This second chart shows the logic operator || evaluation.

Policy X	Policy Y	Policy X || Policy Y
true	true	true
true	false	true
false	true	true
false	false	false

And this third chart shows the logic operator ! evaluation.

Policy X	! Policy X
true	false
false	true

Now let's look at the three operators with respect to our preceding example policy.

Examples: &&

The && operator is referenced under the protocol BGP as an export statement, and both policies, `match-172-16` and `match-172-17`, must return a value of true in order for the evaluated policy to be true:

```
[edit]
jack# show protocols
bgp {
   export ( match-172-16 && match-172-17 );
}
```

Using the test prefix 172.16.100.0/24 to evaluate the && policy expression, the final result is that the prefix is rejected, as listed in this chart:

Prefix:172.16.100.0/24		
Policy match-172-16	Policy match-172-17	&& result
True	False	False

Examples: ||

The || Boolean policy operator is referenced under the protocol BGP as an export statement. Only one of the policies, `match-172-16` or `match-172-17`, must return a value of true in order for the evaluated policy to be true:

```
[edit]
jack# show protocols
bgp {
   export ( match-172-16 || match-172-17 );
}
```

Using the test prefix 172.16.100.0/24 to evaluate the || policy expression, the final result is that the prefix is rejected, as listed here in this chart:

Prefix:172.16.100.0/24		
Policy match-172-16	Policy match-172-17	|| result
True	False	True

Examples: !

The test prefix 172.16.100.0/24 is evaluated using the **!** policy expression, and the final result is that the prefix is accepted:

```
[edit]
jack# show protocols
bgp {
   export ( !match-172-17 );
}
```

And listed here in this chart:

Prefix:172.16.100.0/24	
Policy match-172-17	! result
False	True

Down the Rabbit Hole

Boolean logic doesn't end with simple examples. Advanced expressions can also be created. Between grouping policies and applying algorithmic order of operations the possibilities for policy creating are almost limitless. Take for example:

```
[edit]
jack# show protocols
bgp {
   export ( ( match-172-17 || match-172-17 ) !match-172-18 );
}
```

This policy expression matches prefixes that match `match-172-16` or `match-172-17` but not `match-172-18`.

There are plenty more examples of Boolean arithmetic, but they are beyond the scope of this *Day One* book. Write the editor in chief at dayone@juniper.net and request a title devoted to Boolean arithmetic adventures.

Actions

Policy actions can be divided into two groups: *flow control actions* and *modifier actions*. Flow control actions determine the order in which terms are processed as *each* term is evaluated. Modifier actions manipulate the attributes of the prefixes. Up to this point, this book has only discussed how to match the interesting prefixes and apply basic policy actions. It's time to do something more advanced with all of the identified routes.

TIP The absence of a "from" or "to" statement in a policy is understood by Junos as a *match all* condition. All actions listed under the "then" statement will be applied to all prefixes.

To help explore the actions of policy, let's use the following generic policy called *some-test-policy*:

```
[edit policy-options]
jack# show
```

```
policy-statement some-test-policy {
   term plain-english {
      from {
         route-filter 10.10.0.0/16 exact;
      }
      then {
         metric 10;
         accept;
      }
   }
   term prefix-length-8 {
      from {
         route-filter 10.0.0.0/8 orlonger;
      }
      then {
         metric 20;
         reject;
      }
   }
}
```

Flow Control Actions

Policy flow control is defined as a part of the "then" portion of the configuration and there are terminating actions and flow control actions.

The keywords "accept" and "reject" are considered terminating actions, and terminating actions stop the further processing of the prefixes within the context of the policy. In the example policy *some-test-policy*, the route 10.10.0.0/16 technically matches both terms `plain-english` and `prefix-legth-8`. Since the processing rules for policy follow a top down logic, any prefix that matches the first term, `plain-english`, will be removed from further processing in the policy by the terminating action of `accept`. Likewise, routes other than 10.10.0.0/16 will be passed to the second term, `prefix-length-8`, since they do not match the "from" statement of the term `plain-english`.

Flow control actions don't stop the processing of prefixes; they affect the order of operations within the policy itself. Terms can be skipped, or a prefix can jump to the next policy in a chain to be evaluated by more terms.

NOTE There are also default actions that affect the operational flow of policy when an action statement is not defined. These default actions are discussed in more detail in Chapter 3.

Table 2.4 defines and describes the terminating and flow control actions available in policy.

Table 2.4 Terminating and Flow Control Actions

Flow Control	Description	Terminating or Flow?
accept	Accept the route and propagate it. Stop processing all other terms and/or policies.	Terminating
default-action accept	Accept the route and override the default action for the protocol*.	Terminating
reject	Reject the route and do not propagate it. Stop processing all other terms and/or policies.	Terminating
default-action reject	Reject the route and override the default action for the protocol*.	Terminating
next term	Go to the next term in the policy, if it is the last term then move to the next policy-statement in the chain.	Flow Control
next policy	Go to the next policy in the policy chain. Skip all remaining terms in the current policy-statement.	Flow Control

* Protocols have default actions associated with them. Chapter 3 will describe these default behaviors.

Modifier Actions

Modifier actions are responsible for changing the attribute of the routes. Attributes such as metrics, or adding route tags and BGP communities, are handled by modifier actions. In the example policy *some-test-policy*, the action modifiers shown are specifically for adjusting the metric of the routes. The metric will be set to a value of 10 or 20 depending on which term a prefix matches. Table 2.5 lists some of the common modifiers available in policy, but it is not an all-inclusive list. For that level of thoroughness, see the current Junos *Policy Framework Configuration Guide* at http:www.juniper.net/techpubs/.

Table 2.5 Common Modifier Actions

Modifying Action	Description
as-path-prepend as-path	Used with BGP to prepend one or more instances of an AS to a BGP advertisement.
community (+ | add) [names]	Add a BGP community to the prefix.
community (– | delete) [names]	Delete a BGP community to the prefix.
community (= | set) [names]	Replace the BGP communities on a prefix.

external type metric	Change the OSPF external metric type on a given prefix.
install-nexthop <strict> lsp lsp-name	Selects a next-hop, among many equal cost next-hops, for a given lSP and installs that next-hop of the lSP in the forwarding table.
local-preference value	Change the BGP local preference.
metric (add | subtract) number	Change the metric for a given prefix. For BGP, metric set the MED value.
next-hop (address | discard | next-table routing-table-name | peer-address | reject | self)	Set the next-hop of a given prefix.
origin value	Sets the BGP origin attribute.
preference preference	Changes the route preference (administrative distance) for a particular route.
tag tag	Assign a numeric value in the tag field of an OSPF, ISIS, or RIPv2 route.

Policy Evaluation Logic

Junos processes policy in a logical way, starting with the first policy statement and its first term and continuing in a top down manner until all policy statements and terms are processed, including the default policy. Only when a route matches a term that contains a terminating action does the policy processing stop.

Policy Flow

Policy flows in an ordered manner from top to bottom, working through each term until a match is encountered and a terminating action is applied. Let's view a policy configuration snippet to gain understanding of how a route is processed by policy.

TIP Use the *Day One* books on the Junos CLI to improve your CLI skills so that you can become familiar with, and proficient in, tasks like the reordering of policy terms.

Single Policy Flow Example

Here, the prefix 10.10.10.0/24 is learned via the external BGP neighbor in your network, and on that BGP neighbor is an applied policy called *test-bgp-policy* which contains three terms. Let's evaluate the prefix 10.10.10.0/24 against the policy to demonstrate policy flow:

```
policy-statement test-bgp-policy {
    term 1 {
        from {
            route-filter 10.10.0.0/16 exact accept;
        }
        then {
            metric 25;
            accept;
        }
    }
    term 2 {
        from {
            protocol ospf;
            route-filter 10.10.10.0/24 longer;
        }
        then accept;
    }
    term 3 {
        from {
            protocol bgp;
            route-filter 10.10.0.0/17 orlonger;
        }
        then {
            local-preference 150;
            accept;
        }
    }
}
```

If you evaluate the prefix 10.10.10.0/24 against policy, you can deduce the following. The first term—term 1—is actually tricky, as you have to pay close attention to the terminating action that is placed on the route-filter itself:

```
    term 1 {
        from {
            route-filter 10.10.0.0/16 exact accept;
        }
        then {
            metric 25;
            accept;
        }
    }
```

Term 1's "then" statement is redundant and the metric will not be changed on any routes that match the route filter. Since term 1 does not match our test prefix, term 2 is evaluated next. While the route-filter portion of the match conditions looks like it may match the 10.10.10.0/24 prefix, the match type is referenced as *longer*. This means only prefixes in the 10.10.10.0/24 supernet with a prefix length

of /25 through /32. Next, as stated in the beginning of this exercise, the 10.10.10.0/24 route was received from a BGP neighbor so the match condition `protocol ospf` is problematic:

```
term 2 {
    from {
        protocol ospf;
        route-filter 10.10.10.0/24 longer;
    }
    then accept;
}
```

The route 10.10.10.0/24 has not matched term 1 or term 2, so it continues to term 3 for evaluation. The match conditions are an exact match for the received prefix – it was received from a BGP neighbor and falls into the 10.10.0.0/17 orlonger range – that satisfies a complete match of all the conditions. Two actions are applied to the route. The modifying action `then local-preference` changes the BGP local-preference attribute to the value of 150. The terminating action of `accept` removes the prefix from further policy processing and places it in the routing table:

```
term 3 {
    from {
        protocol bgp;
        route-filter 10.10.0.0/17 orlonger;
    }
    then {
        local-preference 150;
        accept;
    }
}
```

POP QUIZ On which term from the above policy would the following route match be shown in the output?

```
jack@M120-1a> show route protocol ospf

inet.0: 226 destinations, 640 routes (226 active, 0 holddown, 12 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.10.128/29   *[OSPF/10] 01:00:13, metric 2
                > to 172.25.132.2 via ge-2/0/4.101
```

Answer: Term 2. The route was learned via OSPF and is within the range covered by the route filter so it meets both match criteria.

Single Policy Flow Summary

Ninety percent of the policies that you'll come across in the field will be single policies with multiple terms, and routes are evaluated against the policy in a top down order. So everything discussed so far is key to your network operation, and if you are familiar with the policy of other vendors (i.e., route-maps) this logic and syntax makes sense.

But the power of Junos does not stop at single policies because the OS can create *policy chains* by stringing policy statements together. Let's examine.

Policy Chains

Policy chaining is the function by which routes are evaluated through the linking together of several policies in their listed order. Policy chains are useful when assigning specific functions to discrete policies and then sequentially chaining all of the policies together to affect the final selection and manipulation of routes for a given protocol.

MORE? It's worth mentioning policy chains because of the control that Junos policy allows to affect the flow of routing information. Policy chains are an advanced policy topic and this chapter will just scratch the surface of that topic. It's recommended that you review the Junos documentation set to fully understand its capabilities, at http://www.juniper.net/techpubs/.

Figure 2.2 represents a visual example of the flow of a policy chain and you can see how it differs from single policy flow.

To augment Figure 2.2 let's look at an example of how a simple policy chain is tied to a protocol. In the following code snippet, two policy statements are working together to affect how routes from different BGP neighbors are treated, namely `modify-attr-bgp` and `take-action`. Individual policies are chained under a specific protocol's export or import statement. Policy order is important.

```
[edit protocols bgp]
jack# show
export [ modify-attr-bgp take-action ];
```

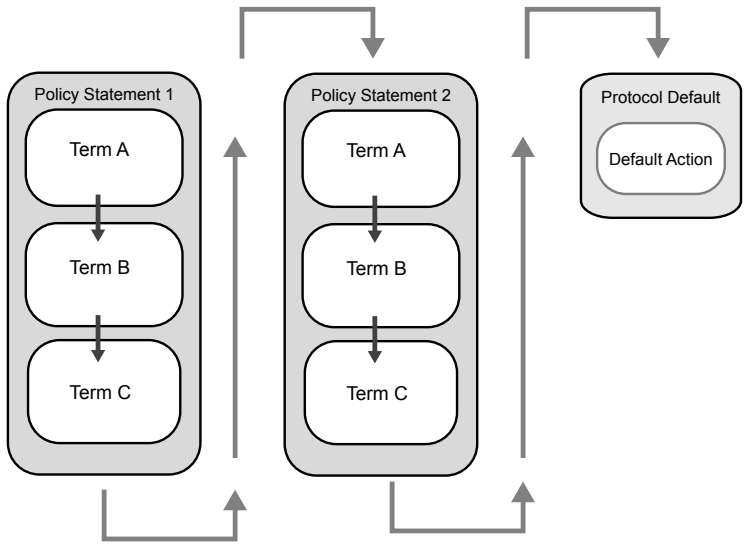


Figure 2.2 Policy Chain Flow

Let's review the two policies to gain a better understanding of what the policy chain does. Notice in the first policy, `modify-attr-bgp`, that the policy actions are used only to modify the attributes of the routes followed by a flow control action. Terminating actions are used for the second policy, `take-action`.

```
[edit policy-options]
jack# show
policy-statement modify-attr-bgp {
    term 1 {
        from neighbor 1.1.1.1;
        then {
            community add vzb;
            next policy;
        }
    }
    term 2 {
        from neighbor 2.2.2.2;
        then {
            community add att;
            next policy;
        }
    }
}
policy-statement take-action {
```

```
    term verizon {
        from community vzb;
        then {
            local-preference 150;
            accept;
        }
    }
    term att {
        from community att;
        then accept;
    }
}
community att members 65000:2222;
community vzb members 65000:1111;
```

What is This Policy Chain Doing?

This policy chain was written as a mutual pair and both policies have to be used together to be effective. The first policy, `modify-attr-bgp`, is responsible for the modification of route attributes as they are learned from two different BGP neighbors. Routes learned from the BGP neighbor 1.1.1.1 will have the BGP community of 65000:1111 added and BGP neighbor 2.2.2.2 will have the BGP community of 65000:2222 added. The flow allows for the community to be added and then the route will be evaluated by the next policy.

The second policy, `take-action`, looks for the community added in the first policy and then takes action on the prefixes. The term *verizon* sets a local preference of 150 to all routes with the community 65000:1111 and then further processing is terminated with the action *accept*. Routes learned from the community of 65000:2222 are accepted as are those with only the community added to the prefix.

Why would a simple action of adding a community to a route require a policy chain? It doesn't. But it does illustrate the modularity of Junos policy, which allows for the creation of policy templates that may be reused in many policy chains and in creating smaller router configurations. Simply stated, this example is meant to demonstrate the function and flow policy chains.

Again, we're just touching the surface of policy chaining, so be sure to follow up on your own after finishing this short book.

Chapter 3

Putting Policy to Work

Now that we have some distance behind us on basic policy definition and creation, Chapter 3 demonstrates the practical aspects of routing policy – marrying policy with protocols. While more discussion and more background theory would build additional understanding, let's go ahead and get started right away with a real world scenario, because hands-on application is the real teacher.

Default Routing Policy and Direction

Up to this point, the mechanics and syntax of policy have been the main subjects of discussion and the fact that policy is used by routing protocols has only been alluded to. It's time to provide you with examples of configuration and demonstrate its effect on the routing of prefixes.

But before beginning configuring policy on routers there are two points you need to review: the default routing policy and the policy direction. Every routing protocol has a default policy that is invoked at the end of all user-defined policy. It is the final policy and it is implicit. Policy direction is also important, and in Junos, there are *import* policies and *export* policies.

The Default Policies

The implicit routing polices are necessary to make the default routing work, otherwise users would need to configure policy every time to make routing work when it is expected to behave in a standard and specific way. For example, it's reasonable to say that most administrators expect BGP to share the BGP routes that a router is aware of. It's the default policies that make this happen.

But default polices can be used to augment user-defined policies, and Table 3.1 lists where the default policy resides in the Junos evaluation chain. Let's quickly discuss the default policy of each.

BGP Default Policy

The BGP default policy is simple: any route learned from a BGP neighbor will be accepted into the routing table and BGP will advertise all BGP learned routes that are active in the routing table with other configured neighbors. Of course, BGP advertising rules apply with respect to internal BGP and external BGP forwarding rules. Policy is required to share other routes learned through other protocols.

Table 3.1 Default Protocol Polices

Protocol	Default Import Policy	Default Export Policy
BGP	Accept all BGP IPv4 routes learned from configured neighbors and import into the inet.0 routing table.	Accept and export active BGP routes.
OSPF/ISIS	Accept all OSPF/ISIS routes and import into the inet.0 routing table. (You cannot override or change this default policy.)	Reject everything. (The protocol uses flooding to announce local routes and any learned routes.)
RIP	Accept all RIP routes learned from configured neighbors and import into the inet.0 routing table.	Reject everything. To export RIP routes, you must configure an export policy for RIP.

OSPF/ISIS Default Policy

OSPF's default policy is slightly more complicated than BGP's; since OSPF uses a link state database (LSDB), OSPF shares routing information apart from the routing table. Link state routing requires that all routers must share native OSPF routes with all other routers and possess an identical link state database. Policy is only used to affect routing information in the routing table - the link state database cannot be modified by policy. OSPF classifies routes into two types: *internally learned* and *externally learned*.

That being said, the shortest calculated path in the LSDB is, by default, imported into the route table. Interfaces configured under the OSPF protocol stanza are also present in the LSDB as native OSPF interfaces. There is no need to create a policy to enable OSPF interface routes. These interface routes are internally learned by OSPF.

Adding routes learned from other protocols to the LSDB is accomplished through an OSPF export policy. If RIP routes need to be shared (also known as redistributed) with OSPF, then those RIP routes that are active in the routing table are exported into the OSPF LSDB. Routes from other protocols, like static, RIP, and BGP, are considered external OSPF prefixes.

RIP Default Policy

RIP learns routes from RIP neighbors, but does not advertise any routes to RIP neighbors. Policy must be explicitly configured to advertise any route, including RIP learned routes, to configured neighbors.

NOTE The default RIP policies illustrate Juniper's ISP-centric history when RIP was a great lightweight protocol for provider-to-customer routing. RIP is a protocol that learns but doesn't share routes in Junos.

The Policy Direction

If you read Table 3.1 carefully you might ask "importing and exporting to what?" And that's a good question. Junos policy is applied in relation to the route table, also known as the routing information base, aka the RIB. Import actions affect the routing information from a specific protocol and neighbor *toward* the route table. Export actions affect what is advertised *to a given neighbor from* the routing table, default policies not withstanding. Figure 3.1 provides a visual example of where import and export polices apply in relation to the individual protocols.

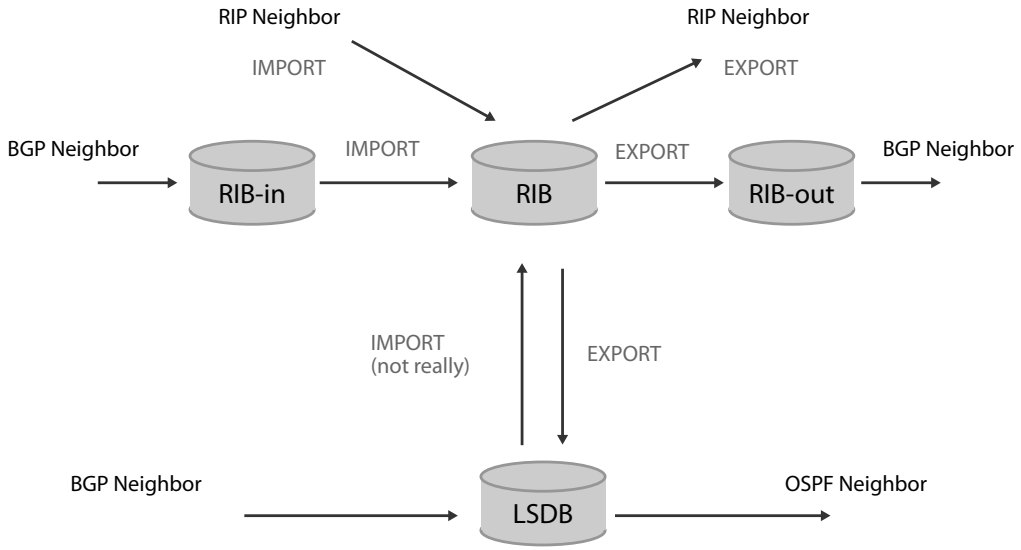


Figure 3.1 Import and Export Policy Directions

Notice how OSPF works. The LSDB is separate from the route table. Policy only affects how routes move between the LSDB and the routing table and not how they move between OSPF neighbors like the other protocols. It is important to point out that BGP utilizes two additional routing tables: RIB-in and RIB-out. RIB-in are the routes received from a BGP neighbor *before* routing policy is applied. RIB-out is the route table *after* policy is applied and just before BGP transmits the prefixes to its neighbor.

MORE? There are two commands that are important to remember for viewing the RIB-in and RIB-out tables: `show route receive-protocol bgp 1.1.1.1` displays the information received for a specific BGP peer *before* the BGP import policy is applied, and to view the routes *after* the BGP export policy is applied, issue the command `show route advertising-protocol bgp 1.1.1.1`.

Applying Policy to Routing Protocols

Let's start demonstrating how policy works so you can practice the concepts discussed in this book. From here on out, this chapter uses the topology shown in Figure 3.2. You will note that there are three routers, West, Central, and East, and each router uses different routing protocols, so our test topology demonstrates OSPF, RIP, and BGP policy.

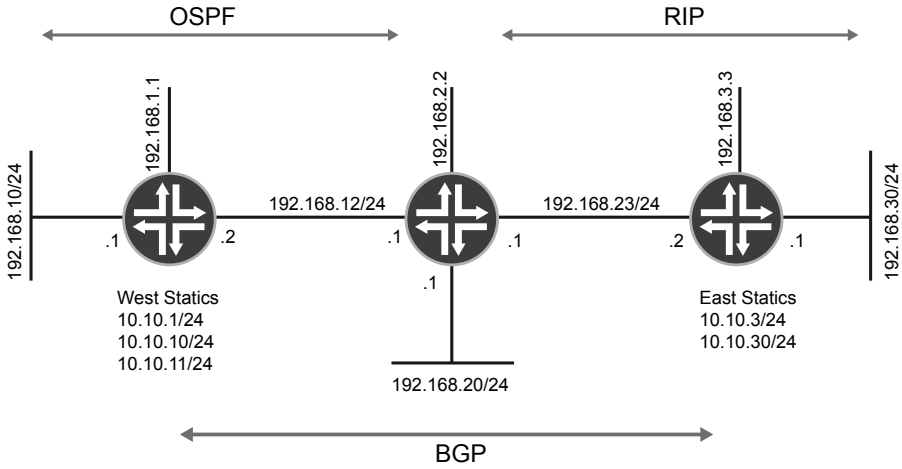


Figure 3.2 Chapter 3's Topology

Network Requirements

The following requirements guide the configurations in this chapter. The basic protocol configurations have already been completed, and from here on out the focus will be strictly on the policy required to achieve the desired results dictated by the requirements:

1. Advertise the static route 10.10.1.0/24 on the West router into OSPF.
2. Advertise the LAN interface, 192.168.20.0/24, on the Central router into RIP with a metric of 10, and into OSPF as an OSPF external type 1.
3. Advertise the Central router's loopback interface into RIP.
4. Advertise the East router's LAN, loopback interface, and 10.10.3.0/24 static route into RIP.
5. On the Central router, advertise West's loopback address from OSPF to RIP, and East's loopback address from RIP into OSPF. Additionally, the transit interface between the East and Central must be advertised to the OSPF neighbor and the transit interface between West and Central must be advertised into RIP.
6. Establish a BGP session between the West and East routers using the loopback addresses and ASN 65000.
7. Advertise the static route 10.10.30.0/24 from the East router to the West router using BGP with a community of 65000:3333. Advertise the LAN address, 192.168.30.0/24, using BGP with a community of 65000:4444.
8. Advertise the static routes 10.10.10.0/24 and 10.10.11.0/24 in BGP from the West router to the East router using a single route-filter statement. Utilize a prefix list to advertise the LAN address 192.168.10.0/24 to the East router.
9. On the West router, strip the received community from the East router's LAN advertisement.

Initial West Router Policy Configuration

First, let's advertise the static route 10.10.1.0/24 on the West router into OSPF.

Quickly verify the West router's configuration of OSPF and neighbor state, so you can begin tackling the first requirement for the policy:

```
[edit]
jack@west# show protocols
ospf {
   area 0.0.0.0 {
      interface fe-0/0/5.0;
      interface fe-0/0/7.0;
      interface lo0.0 {
         passive;
      }
   }
}

[edit]
jack@west# run show ospf neighbor
Address        Interface    State     ID             Pri  Dead
192.168.12.1    fe-0/0/7.    Full      192.168.2.2      128    33
```

The configuration and neighbor state look good. The requirement states that the 10.10.1.0/24 static route must be advertised into OSPF. A check of the routing table verifies that this route is available and active:

```
[edit]
jack@west# run show route 10.10.1.0/24

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.1.0/24       *[Static/5] 01:46:09
                    Discard
```

The policy to accomplish this task is simple to configure – create the policy `ospf-export-policy` to advertise the static route 10.10.1.0/24 into the LSDB:

```
[edit]
jack@west# show policy-options
policy-statement ospf-export-policy {
   term advertise-static {
      from {
         protocol static;
         route-filter 10.10.1.0/24 exact;
      }
      then accept;
   }
}
jack@west# show protocols
ospf {
   export ospf-export-policy;
   area 0.0.0.0 {
```

```
        interface fe-0/0/5.0;
        interface fe-0/0/7.0;
        interface lo0.0 {
            passive;
        }
    }
}
```

Verify the results on the Central router after committing the configuration on the West router. The 10.10.1.0/24 shows up as an OSPF external route:

```
jack@central> show route 10.10.1.0/24

inet.0: 15 destinations, 15 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.1.0/24       *[OSPF/150] 00:00:51, metric 0, tag 0
                  > to 192.168.12.2 via fe-0/0/7.0
```

Okay, everything looks good so far.

Initial Central Router Policy Configuration

Now let's advertise the LAN interface, 192.168.20.0/24, on the Central router into RIP with a metric of 10, and into OSPF as an OSPF external type 1.

Requirements 2 and 3 are to be accomplished on the Central router. For these steps, you need to configure a policy for both RIP and OSPF. The LAN interface address, 192.168.20.0/24, must be advertised via RIP with a metric of 10 and into OSPF as an external type 1 (the default OSPF external type is an external type 2). Then the Central router's loopback address needs to be advertised into both RIP and OSPF.

MORE? The default OSPF external metric type is type 2. Type 2 routes have fixed metrics that do not increment as the route propagates from router to router. The cost is determined by the cost from the ASBR to the destination network, which is usually a static route, with a typical cost of 1. OSPF external type 1 has a variable metric that accumulates the link cost as the route propagates away from the ABSR.

Best practice calls for you to verify the Central router to ensure that the protocols are configured and functioning:

```
[edit]
jack@central# run show ospf neighbor
Address       Interface    State    ID            Pri  Dead
192.168.12.2   fe-0/0/7.0    Full     192.168.1.1     128    38

[edit]
jack@central# run show rip neighbor
                    Source         Destination    Send   Receive   In
Neighbor    State  Address     Address      Mode   Mode     Met
--------    -----  -------     -----------   ----   -------  ---
fe-0/0/6.0   Up 192.168.23.1    224.0.0.9     mcast  both     1
```

OSPF was verified during the previous step and you've seen the effect of routing policy on the West router, so now let's configure OSPF policy on the Central router:

```
[edit]
jack@central# show policy-options
policy-statement ospf-export-policy {
   term adv-lan-address {
      from {
         protocol direct;
         route-filter 192.168.20.0/24 exact;
      }
      then {
         external {
            type 1;
         }
         accept;
         }
   }
}

[edit]
jack@central# show protocols
ospf {
   export ospf-export-policy;
   area 0.0.0.0 {
      interface fe-0/0/7.0;
      interface lo0.0;
   }
}
rip {
   group central {
      neighbor fe-0/0/6.0;
   }
}
```

With OSPF configured and policy applied, you can check the effect on the LSDB on the Central router. Let's look at the LSDB on the Central router since the LSDB operates autonomously from the routing table and will immediately show the effects of policy:

```
 [edit]
jack@central# run show ospf database external extensive
    OSPF AS SCOPE link state database
 Type      ID            Adv Rtr         Seq      Age  Opt  Cksum  Len
Extern   10.10.1.0      192.168.1.1     0x80000002  1148  0x22 0xdf51  36
  mask 255.255.255.0
  Topology default (ID 0)
    Type: 2, Metric: 0, Fwd addr: 0.0.0.0, Tag: 0.0.0.0
  Aging timer 00:40:51
  Installed 00:19:05 ago, expires in 00:40:52
  Last changed 01:06:29 ago, Change count: 1
Extern  *192.168.20.0    192.168.2.2     0x80000002   677  0x22 0xc680  36
  mask 255.255.255.0
  Topology default (ID 0)
    Type: 1, Metric: 0, Fwd addr: 0.0.0.0, Tag: 0.0.0.0
  Gen timer 00:38:43
  Aging timer 00:48:43
  Installed 00:11:17 ago, expires in 00:48:43, sent 00:11:17 ago
  Last changed 00:11:17 ago, Change count: 2, Ours
```

As you can see, two external routes appear in the LSDB. The first route is from the West router and you can see that it is an external type 2. The second route is from the Central router. Notice the * that shows that the route is an external type 1. In order to see the metric increase, you need to view the route on a different router, like the West router:

```
[edit]
jack@west# run show route 192.168.20

inet.0: 13 destinations, 13 routes (13 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.20.0/24    *[OSPF/150] 00:19:48, metric 1, tag 0
                 > to 192.168.12.1 via fe-0/0/7.0
```

If you remember our list of requirements, they also instruct us to configure RIP policy on the Central router. So let's create a policy to advertise the LAN network into RIP with a metric of 10. Remember, by default, RIP only receives RIP routes, but never sends anything: policy is required:

```
 [edit]
jack@central# show policy-options
policy-statement rip-export-policy {
   term adv-lan-address {
      from {
         protocol direct;
         route-filter 192.168.20.0/24 exact;
      }
      then {
         metric 10;
         accept;
      }
   }
}

jack@central# show protocols
ospf {
   export ospf-export-policy;
   area 0.0.0.0 {
      interface fe-0/0/7.0;
      interface lo0.0;
   }
}
rip {
   group central {
      export rip-export-policy;
      neighbor fe-0/0/6.0;
   }
}
```

With a quick look on the East router, Central's RIP neighbor, you can verify that the policy is effective:

```
[edit]
jack@east# run show route protocol rip

inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.20.0/24    *[RIP/100] 00:10:29, metric 11, tag 0
                  > to 192.168.23.1 via fe-0/0/7.0
224.0.0.9/32       *[RIP/100] 01:34:49, metric 1
                    MultiRecv
```

The route is being received via RIP and the metric is shown as 11, which is a hop cost of 1 plus the metric cost of 10 that was set in policy.

Advertise the Central Router's Loopback linterface into RIP

The third listed requirement is to advertise the Central router's loopback address into RIP. We have several choices to make at this point and there are several options available to accomplish this task. We can create a new policy and create a policy chain, but that seems a bit excessive. Rereading all of the configuration tasks, nothing seems to require a policy chain now or in the future. We can add a route filter to the existing policy. However, that policy term has an action modifier to set the metric to 10, which is not what we want to happen either. The best course of action is to use the existing policy and add a new term.

```
policy-statement rip-export-policy {
    term adv-lan-address {
        from {
            protocol direct;
            route-filter 192.168.20.0/24 exact;
        }
        then {
            metric 10;
            accept;
        }
    }
    term adv-loopback {
        from {
            protocol direct;
            route-filter 192.168.2.2/32 exact accept;
        }
    }
}
```

The term *adv-loopback* has been added to the policy *rip-export-policy*. An important note about the configuration is that any terms added to an existing policy are appended to the end of the term list. Order is important in policy, but in this case order does not produce adverse effects in the processing of the policy. Also, a terminating action was added to the end of the route-filter, which negates the need for a "then" statement.

The updated route table as viewed from the West router.

```
[edit]
jack@east# run show route protocol rip

inet.0: 13 destinations, 13 routes (13 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
192.168.2.2/32     *[RIP/100] 00:14:35, metric 2, tag 0
                 > to 192.168.23.1 via fe-0/0/7.0
192.168.20.0/24    *[RIP/100] 00:27:42, metric 11, tag 0
                 > to 192.168.23.1 via fe-0/0/7.0
224.0.0.9/32       *[RIP/100] 01:52:02, metric 1
                    MultiRecv
```

Initial East Router Policy Configuration

Now it's time to advertise the East router's LAN, loopback interface, and 10.10.3.0/24 static route into RIP.

The East router, according to requirement four, needs to advertise the directly attached networks for the loopback interface, LAN interface, and static route 10.10.3.0/24 into RIP.

```
[edit]
jack@east# show policy-options
policy-statement rip-export-policy {
   term adv-networks {
      from {
         protocol [ static direct ];
         route-filter 192.168.3.3/32 exact;
         route-filter 192.168.30.0/24 exact;
         route-filter 10.10.30.0/24 exact;
      }
      then accept;
   }
}
```

The policy `rip-export-policy` on the East router is a good example of different Boolean operations in a single policy. There are two types of routes, *connected* and *static*, listed in the requirements for this router, and there are two directly connected routes and a single static route that must be addressed.

The first condition for a match is based on source protocol. The policy will match routes that are either static routes or directly connected interfaces. The protocol condition and the route filters narrow the selection criteria by only accepting routes that match both conditions. Route filters are the longest match and are not an ordered, top-down match like an access list.

If you look at the East router's upstream neighbor, Central, you can verify the policy is exporting only the three specified prefixes:

```
[edit]
jack@central# run show route protocol rip

inet.0: 18 destinations, 18 routes (18 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.30.0/24      *[RIP/100] 00:00:18, metric 2, tag 0
                  > to 192.168.23.2 via fe-0/0/6.0
192.168.3.3/32     *[RIP/100] 00:00:18, metric 2, tag 0
                  > to 192.168.23.2 via fe-0/0/6.0
192.168.30.0/24    *[RIP/100] 00:00:18, metric 2, tag 0
                  > to 192.168.23.2 via fe-0/0/6.0
224.0.0.9/32       *[RIP/100] 00:39:12, metric 1
                    MultiRecv
```

So far, everything is working as expected. Another helpful Junos capability is the ability to view the outgoing RIP advertisements. The "neighbor" IP address at the end of the following command is the local outgoing interface IP address and can be very helpful, especially without administrative access to the upstream router:

```
[edit]
jack@east# run show route advertising-protocol rip 192.168.23.2

inet.0: 13 destinations, 13 routes (13 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.30.0/24      *[Static/5] 02:16:20
                    Discard
192.168.3.3/32     *[Direct/0] 02:16:20
                  > via lo0.0
192.168.30.0/24    *[Direct/0] 02:14:20
                  > via fe-0/0/5.0
```

This requirement is complete and the desired effects have been accomplished with the necessary policy.

Protocol Redistribution on the Central Router

Next up is to advertise West's loopback address from OSPF to RIP and East's loopback address from RIP into OSPF, on the Central router. Additionally, the transit interface between East and Central must be advertised to the OSPF neighbor, and the transit interface between West and Central must be advertised into RIP.

Before moving into BGP policy configuration, you need to share routing information between OSPF and RIP. BGP typically requires an

underlying IGP to establish peering sessions and resolve next-hop addresses in BGP advertisements. The Central router is a pivotal router in the topology and is the only router that participates in both IGP routing protocols.

To satisfy the fifth requirement, you need to modify the existing OSPF and RIP policies on the Central router to perform a mutual redistribution of routing information. The West router needs routing knowledge of the East router's loopback address and the transit interface between the Central router and the East router. Inversely, the East router needs routing knowledge of the West router's loopback address and the Central and West router transit interface.Let's look at the OSPF policy, with the necessary updates, first:

```
[edit]
jack@central# show policy-options policy-statement ospf-export-policy
term adv-lan-address {
    from {
        protocol direct;
        route-filter 192.168.20.0/24 exact;
    }
    then {
        external {
            type 1;
        }
        accept;
    }
}
term for-bgp {
    from {
        protocol [ rip direct ];
        route-filter 192.168.3.3/32 exact;
        route-filter 192.168.23.0/24 exact;
    }
    then accept;
}
```

A new term was added to the OSPF policy, `ospf-export-policy`, named `for-bgp`. This simple term takes the two networks listed in the requirements and exports them to the OSPF LSDB.

MORE? It's easy to forget that the OSPF export statement is used for populating the OSPF LSDB with non-OSPF prefixes. Unlike BGP and RIP, the policy is *not* toward a configured neighbor. The export policy direction is from the route-table toward the LSDB. The LSDB handles the route propagation between OSPF routers.

```
[edit]
jack@central# show policy-options policy-statement rip-export-policy
term adv-lan-address {
    from {
        protocol direct;
        route-filter 192.168.20.0/24 exact;
    }
    then {
        metric 10;
        accept;
    }
}
term loopback {
    from {
        protocol direct;
        route-filter 192.168.2.2/32 exact accept;
    }
}
term for-bgp {
    from {
        protocol [ ospf direct ];
        route-filter 192.168.1.1/32 exact;
        route-filter 192.168.12.0/24 exact;
    }
    then accept;
}
```

The RIP policy is similar to the OSPF policy. A new term, `for-bgp`, was appended to the existing RIP policy, `rip-export-policy`, to share the required routes with the East router.

Here are West's current OSPF routes:

```
[edit]
jack@west# run show route protocol ospf

inet.0: 15 destinations, 15 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.2/32     *[OSPF/10] 02:56:04, metric 1
                  > to 192.168.12.1 via fe-0/0/7.0
192.168.3.3/32     *[OSPF/150] 00:13:46, metric 2, tag 0
                  > to 192.168.12.1 via fe-0/0/7.0
192.168.20.0/24    *[OSPF/150] 02:01:02, metric 1, tag 0
                  > to 192.168.12.1 via fe-0/0/7.0
192.168.23.0/24    *[OSPF/150] 00:13:46, metric 0, tag 0
                  > to 192.168.12.1 via fe-0/0/7.0
224.0.0.5/32       *[OSPF/10] 02:58:53, metric 1
                    MultiRecv
```

And East's current RIP routes:

```
[edit]
jack@east# run show route protocol rip

inet.0: 15 destinations, 15 routes (15 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.1.1/32     *[RIP/100] 00:14:50, metric 2, tag 0
                  > to 192.168.23.1 via fe-0/0/7.0
192.168.2.2/32     *[RIP/100] 01:22:52, metric 2, tag 0
                  > to 192.168.23.1 via fe-0/0/7.0
192.168.12.0/24    *[RIP/100] 00:14:50, metric 2, tag 0
                  > to 192.168.23.1 via fe-0/0/7.0
192.168.20.0/24    *[RIP/100] 01:35:59, metric 11, tag 0
                  > to 192.168.23.1 via fe-0/0/7.0
224.0.0.9/32       *[RIP/100] 00:44:07, metric 1
                    MultiRecv
```

The requirement has been satisfied by the recent configuration updates. The specified routing information is now available to the East and West routers. BGP can now be configured thanks to the shared reachability information.

BEST PRACTICE

This *Day One* book uses a simple network topology. In real networks, single routers seldom perform mutual route redistribution between protocols. Multiple routers sharing routing information between protocols can create routing loops or blackholes thanks to differing protocol preferences. Although an explanation of how to do so is beyond the scope of this book, it is recommended that you use the tag attribute to prevent the reintroduction of routing information back into the originating protocol.

BGP Configuration on East-West Routers

Our next requirement, number six on the list of nine, is to establish a BGP session between the West and East routers using the loopback addresses and ASN 65000.

In order to use BGP policy, the routers must have active BGP sessions. According to the requirement, a BGP session must be established between the East and West routers. Using the new routing information from the previous step, let's configure BGP between the loopback addresses of the East and West Routers and use ASN 65000.

Here's West's BGP configuration:

```
[edit]
jack@west# show protocols bgp
group east-west {
   type internal;
   local-address 192.168.1.1;
   neighbor 192.168.3.3;
}
```

And East's BGP configuration:

```
[edit]
jack@east# show protocols bgp
group east-west {
   type internal;
   local-address 192.168.3.3;
   neighbor 192.168.1.1;
}
```

Let's verify that the BGP session is established between the East and West routers:

```
[edit]
jack@west# run show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table         Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0               0          0          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
192.168.3.3          65000     3         4 0       0         33 0/0/0/0           0/0/0/0
```

As you can see, BGP is established. There is no exchange of routing information at this time since there is a lack of policy. The default export policy for BGP is only to advertise other active BGP routes. The `active/received/accepted/damped` field shows 0/0/0/0. The remaining tasks will focus on BGP policy.

BEST PRACTICE

It is strongly suggested that you be as specific as possible in policy without requiring finely-detailed terms for every single route. To do this, try reducing the policy down to the least common denominator. Route filters are a great way to narrow down to a specific supernet, but also use additional controls like a protocol match condition.

East Router BGP Policy Configuration

We're already at the seventh requirement on our list! How time flies. Now let's advertise the static route 10.10.30.0/24 from the East router to the West router using BGP with a community of 65000:3333. Advertise the LAN address, 192.168.30.0/24, using BGP with a community of 65000:4444.

The goal of this task is to configure the East router's BGP policy to advertise the static route 10.10.30.0/24 with the community of 65000:3333 and the LAN address 192.168.30.0/24 with a community of 65000:4444.

The first thing that must be done is to define the communities. Communities referenced in policy are user variables. In Junos policy, communities cannot be added to policy as native numeric variables, they must be defined under `policy-options`.

It is recommended that you define the community variables first. This way the communities can be tab-completed by the CLI when creating the policy. The chance of misconfiguration, or a configuration that will not commit, could occur if the community is defined after the policy is created.

Remember, define the variables – then you can reference the variables. Like so:

```
community east-lan members 65000:4444;
community east-static members 65000:3333;
```

Next, you can construct the policy to meet the requirements:

```
jack@east# show policy-options
policy-statement bgp-export-policy {
   term adv-static {
      from {
         protocol static;
         route-filter 10.10.30.0/24 exact;
      }
      then {
         community add east-static;
         accept;
      }
   }
   term adv-lan {
      from {
         protocol direct;
         route-filter 192.168.30.0/24 exact;
      }
```

```
        then {
            community add east-lan;
            accept;
        }
    }
}
```

Finally, the policy is applied to the BGP group `east-west` as an export policy:

```
[edit]
jack@east# show protocols bgp
group east-west {
    type internal;
    local-address 192.168.3.3;
    export bgp-export-policy;
    neighbor 192.168.1.1;
}
```

If you look at the West router, you can see that it is receiving the two prefixes from the East router:

```
[edit]
jack@west# run show route protocol bgp

inet.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.30.0/24      *[BGP/170] 00:01:12, localpref 100, from 192.168.3.3
                    AS path: I
                  > to 192.168.12.1 via fe-0/0/7.0
192.168.30.0/24    *[BGP/170] 00:01:12, localpref 100, from 192.168.3.3
                    AS path: I
                  > to 192.168.12.1 via fe-0/0/7.0
```

To verify that the communities were applied to the BGP advertisements and that the West router is receiving the communities, you have to take a detailed look at the prefixes. For the sake of brevity in this book, let's look at a single route:

```
[edit]
jack@west# run show route protocol bgp 10.10.30.0/24 detail

inet.0: 17 destinations, 17 routes (17 active, 0 holddown, 0 hidden)
10.10.30.0/24 (1 entry, 1 announced)
        *BGP    Preference: 170/-101
                Next hop type: Indirect
                Next-hop reference count: 6
```

```
Source: 192.168.3.3
Next hop type: Router, Next hop index: 555
Next hop: 192.168.12.1 via fe-0/0/7.0, selected
Protocol next hop: 192.168.3.3
Indirect next hop: 14c43fc 262142
State: <Active Int Ext>
Local AS: 65000 Peer AS: 65000
Age: 3:05      Metric2: 2
Task: BGP_65000.192.168.3.3+179
Announcement bits (2): 0-KRT 5-Resolve tree 1
AS path: I
Communities: 65000:3333
Accepted
Localpref: 100
Router ID: 192.168.3.3
```

You can see the communities in boldface. This task is complete.

West Router BGP Policy Configuration

Next on our list is to advertise the static routes 10.10.10.0/24 and 10.10.11.0/24 in BGP from the West router to the East router using a single route-filter statement. Utilize a prefix-list to advertise the LAN address 192.168.10.0/24 to the East router.

The final requirements for this section are completed on the West router. One goal is to advertise the two static routes, 10.10.10.0/24 and 10.10.11.0/24 , using a single route filter. The LAN network 192.168.10.0/24 must be advertised from BGP using a prefix-list. Additionally, the community needs to be stripped off of the East router's LAN network advertisement. Our policy configuration will begin with an export policy to advertise the networks defined in the requirements, and then finish with an import policy to strip the community off of the advertisement.

BEST PRACTICE It is recommended that you define the user variables that will be referenced in policy before creating the policy.

Let's follow best practice and define the user variables:

```
[edit]
jack@west# show policy-options
prefix-list lan-addresses {
   192.168.10.0/24;
}
community all members *:*;
```

Now create the policy required to advertise the required prefixes:

```
[edit]
jack@west# show policy-options
prefix-list lan-addresses {
   192.168.10.0/24;
}
policy-statement bgp-export-policy {
   term adv-statics {
      from {
         protocol static;
         route-filter 10.10.10.0/23 longer;
      }
      then accept;
   }
   term adv-lan-address {
      from {
         protocol direct;
         prefix-list lan-addresses;
      }
      then accept;
   }
}
```

Notice the use of a single route °filter to match both static routes, 10.10.10.0/24 and 10.10.11.0/24. The match type of `longer` only matches on prefixes that are more specific than a /23. Other match types could be used to accomplish the same results.

Let's verify the effectiveness of the policy by checking the received BGP routes on the East router:

```
[edit]
jack@east# run show route protocol bgp

inet.0: 18 destinations, 18 routes (18 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.10.0/24      *[BGP/170] 00:05:53, localpref 100, from 192.168.1.1
                    AS path: I
                  > to 192.168.23.1 via fe-0/0/7.0
10.10.11.0/24      *[BGP/170] 00:01:22, localpref 100, from 192.168.1.1
                    AS path: I
                  > to 192.168.23.1 via fe-0/0/7.0
192.168.10.0/24    *[BGP/170] 00:05:53, localpref 100, from 192.168.1.1
                    AS path: I
to 192.168.23.1 via fe-0/0/7.0
```

Strip the Community from the East Router's LAN Advertisement

Finally our last requirement is due on the West router, to strip the received community from the East router's LAN advertisement

The East router is receiving the three routes via BGP identified in the requirements section. To remove the community from the East router LAN prefix on the West router you create an import policy. The policy `bgp-import-policy` is shown here and applied to the BGP group `east-west`:

```
[edit]
jack@west# show policy-options
policy-statement bgp-import-policy {
    term strip-community {
        from {
            protocol bgp;
            route-filter 192.168.30.0/24 exact;
        }
        then {
            community delete all;
            accept;
        }
    }
    term bgp {
        from protocol bgp;
        then accept;
    }
}
[edit]
jack@west# show protocols bgp
group east-west {
    type internal;
    local-address 192.168.1.1;
    import bgp-import-policy;
    export bgp-export-policy;
    neighbor 192.168.3.3;
}
```

There are two commands that are useful to show the effects of this policy.

First, if you look at the route table for the 192.168.30.0/24 route, you should see the absence of any community. The RIB-in should still see the incoming prefix with the community attached. Remember from the policy direction discussion earlier in this chapter that BGP has a RIB-in

and RIB-out table that contains the prefixes *before* policy is applied. This is extremely useful for troubleshooting when you don't have administrative control of a neighboring router. So from the route table on the West router, issue the following command:

```
[edit]
jack@west# run show route 192.168.30.0/24 detail

inet.0: 18 destinations, 18 routes (18 active, 0 holddown, 0 hidden)
192.168.30.0/24 (1 entry, 1 announced)
       *BGP    Preference: 170/-101
              Next hop type: Indirect
              Next-hop reference count: 6
              Source: 192.168.3.3
              Next hop type: Router, Next hop index: 555
              Next hop: 192.168.12.1 via fe-0/0/7.0, selected
              Protocol next hop: 192.168.3.3
              Indirect next hop: 14c43fc 262142
              State: <Active Int Ext>
              Local AS: 65000 Peer AS: 65000
              Age: 55:59     Metric2: 2
              Task: BGP_65000.192.168.3.3+179
              Announcement bits (2): 0-KRT 5-Resolve tree 1
              AS path: I
              Accepted
              Localpref: 100
              Router ID: 192.168.3.3
```

And the second command is issued from the RIB-in on the West router:

```
[edit]
jack@west# run show route receive-protocol bgp 192.168.3.3 192.168.30.0/24 detail

inet.0: 18 destinations, 18 routes (18 active, 0 holddown, 0 hidden)
* 192.168.30.0/24 (1 entry, 1 announced)
    Accepted
    Nexthop: 192.168.3.3
    Localpref: 100
    AS path: I
    Communities: 65000:4444
```

Summary

The application of policy available in Junos is a powerful tool for sharing network information. Policy is the universal way, across protocols, to advertise and filter the routes shared between routers. Consistent implementation limits the learning curve and accelerates the deployment of routers in the network.

TIP Policy in the Junos OS is universal in its implementation, unlike other networking vendors where the command set is fragmented between distribute-lists, access-lists, route-maps, and the redistribute syntax.

Testing Policy

In the real world, it is difficult to make changes to production routers, especially to routing policy that may trigger convergence, outages, or blackholes. Junos has another tool up its sleeve that is helpful for testing a policy before it is applied to a protocol.

The command is `test policy`. The two variables that must be supplied to the command syntax are the *policy name* and a *destination prefix* that you want to test against the policy, as shown here with the CLI help prompt:

```
jack@west> test policy bgp-export-policy ?
Possible completions:
  <prefix>           Destination prefix
```

WARNING! The test command only works with routes that are active in the routing table (inet.0). If a route is non-existent, invalid, or rejected by applied policy then the test policy command will not return a result.

To demonstrate the test policy command, let's use the West router and and the existing policy `bgp-export-policy`. The first test will verify the prefix 10.10.10.0/24 against the policy:

```
jack@west> test policy bgp-export-policy 10.10.10.0/24

inet.0: 18 destinations, 18 routes (18 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.10.10.0/24      *[Static/5] 06:56:02
                   Discard

Policy bgp-export-policy: 1 prefix accepted, 0 prefix rejected
```

An active match is displayed as an accepted prefix. This means there is a positive match in the policy with the terminating action of `accept`:

```
jack@west> test policy bgp-export-policy 11.11.11.0/24

Policy bgp-export-policy: 0 prefix accepted, 0 prefix rejected

jack@west>
```

If a prefix does not match the policy —meaning no terminating action is applied— then the match count will show no prefixes accepted or rejected. Only if a terminating action of `reject` is present in a policy will the rejected prefix count increment.

TIP Along with testing individual routes against policy, you can submit the entire active routing table by specifying 0/0 as the destination prefix, which is extremely useful in validating things like external BGP advertisement policies when you are interested in understanding which routes may be advertised prior to activating a given policy.

Chapter 4

Firewall Filter Configuration

Firewall filters share a common feel and syntax with Junos policy, but there are important distinctions.

The first thing to note is that firewall filters are used to *affect traffic transiting* the router. Policy, if you remember from the previous chapters, is used to affect the learning and sharing of prefixes in relation to routing protocols. Firewall filters are stateless packet filters.

You may also be familiar with an industry term used to describe firewall filters: *access control list* (ACL). This chapter covers the syntax of firewall filters by breaking down the core components: *match conditions*, *policing*, and *firewall actions*, and concludes with a discussion of the evaluation logic used for the processing of packets within a firewall filter. Along the way the chapter will point out other key distinctions in tandem with syntax examples.

NOTE Firewall filters are compiled and added to the forwarding table allowing the filters to operate at line rate. Juniper hardware can also discard packets at line rate without impacting the forwarding performance. Try that with your other network vendors' hardware.

Firewall Filter Syntax

In order to properly construct a firewall filter, you must observe a few requirements. First, firewall filters are created under a specific protocol family like IPv4, for example. Second, filter terms are used to aggregate match conditions to the corresponding packet filtering actions.

Now, let's dig into the components of properly constructed firewall filters. Here's a sample firewall filter syntax (explanations of the components of the filter to follow):

```
[edit firewall]
  family family-name {
    filter filter-name {
        accounting-profile name;
        interface-specific;
        physical-interface-filter;
        term term-name {
           filter filter-name;
           from {
              match-conditions;
           }
           then {
           action;
           action-modifiers;
```

```
            }
        }
    }
  }
}
```

The Protocol Family

The protocol family is used to determine which packet headers a Junos device inspects to apply the filter conditions and actions. The protocol family is defined with the `family <family-name>` statement. Junos supports the following protocol families when constructing firewall filters.

Protocol Family	Junos Firewall Family-name
IPv4	inet
IPv6	inet6
MPLS	mpls
VPLS	vpls
Circuit Cross Connects	ccc
Bridge (aka vlan) **MX only	bridge

Filter Name

Enabling users to define names for policies, filters, and terms differentiates Junos from other network operating systems. Firewall filters are another element in Junos that is user defined. The named filter is the top-most container that represents a given stateless packet filter. It is the filter name that will be applied on interfaces enforcing the filtering policy.

Terms

Just as in Junos policy, terms are used to group match conditions so that a specific filter action can be applied to the packet. Multiple match conditions can exist under a single term, but only a single final action may be applied per term. Terms are user-defined variables.

BEST PRACTICE

Firewall terms can be ordered and reordered without deleting the firewall filter, editing the filter, and pasting the filter back into the configuration. Use the `insert` command to reorder terms.

Terms can be similar to the line numbers of an ACL. However, they are ultimately much more powerful than an ACL. In real world environments, traditional ACLs can be consolidated into just a few terms, thus shortening long configuration files and identifying intended functions through a user-defined term name by grouping similar actions together.

WARNING

The default action for a term is to *accept*, or permit a packet matching the criteria in the "from" statement. This differs from the default action for a filter, which is an explicit *reject-all*, dropping all traffic that does not match any terms within a firewall filter.

Filter Application

Defining a firewall filter is only part of the necessary configuration required to make it work because the firewall filter must be applied elsewhere in the Junos configuration. Firewall filters may be applied to:

- Physical Interfaces
- Logical Interfaces
- Routing Interfaces
- Routing Instances

Direction is also important in the application of a firewall filter. The firewall is evaluated based on the applied direction. To have the filter evaluate a packet entering an interface, the packet filter must be applied as an *input filter*. If you wish for a packet to be evaluated as it leaves an interface, then the filter is applied as an *output filter*.

Filters can be applied as a *chain of filters*, similar to a policy chain, by using input-list or output-list. This is helpful if you have the standard filter that must be applied to all interfaces and additional filters for a given enforcement point.

Match Criteria

The match criteria in firewall filters are determined by the `from` statement under a given term. Just about any component of an IP packet header may be used to match upon. The match criteria in firewall filters are more straightforward than the matching criteria of policy. The criteria are slightly different between the protocol families, like IPv4 and IPv6, as the header information is different between the protocols.

MORE? This *Day One* book only focuses on the IPv4 protocol family in this Day One book. For more information on the match conditions for other protocol families please reference the *Firewall Filter and Policer Configuration Guide* in the current Junos release documentation suite at http://www.juniper.net/techpubs/.

But there are also common match conditions that may be used to facilitate packet matching for all protocol types, namely:

- Numeric and Text Value Matching
- Prefix Matching
- Bit-field Matching

Let's review each of the common match conditions before focusing on IPv4 protocol matching.

Numeric and Text Value Matching

A *numeric matching* is based on a number and a *text matching* uses a text synonym to represent a numeric value. Matches can be configured as a *single value* or as a *range of values*. Values may also be combined in a *list*.

Single Value Example:

```
source-port 80; source-port http;
```

Range of values:

```
Destination-port 30000-39999;
```

List of values:

```
Source-port [ http https 25 53 1812-1813]
```

Note: Values in a list are treated as OR.

Table 4.1 Common Text Values and Their Associated Numeric Value:

Used For	Text Synonym with Numeric Values
Ports	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), ldp (646), 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 (49), tacacs-ds (65), talk (517), telnet (23), tftp (69), timed (525), who (513), or xdmcp (177)
Class of Service (DSCP)	af11 (10), af12 (12), af13 (14) af21 (18), af22 (20), af23 (22) af31 (26), af32 (28), af33 (30) af41 (34), af42 (36), af43 (38) ef (46)
ICMP-code	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	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)
Protocol Numbers	ah (51), dstopts (60), egp (8), esp (50), fragment (44), gre (47), hop-by-hop (0), icmp (1), icmp6 (58), icmpv6 (58), igmp (2), ipip (4), ipv6 (41), ospf (89), pim (103), rsvp (46), sctp (132), tcp (6), udp (17), or vrrp (112)

Prefix Matching

Prefix matching is typically the core match condition for most packet filters. Most firewall filters are concerned with allowing or blocking packets that originate from, or are destined for, a specific network. Junos firewall filters provide several methods for matching prefixes.

TIP Prefix matching in Junos uses the network address followed by the prefix length. This differs from other vendor's packet filter implementations that rely on the use of a prefix followed by the subnet mask.

Single prefixes:

```
source-address 192.168.0.0/16;
```

Multiple prefixes:

```
source-address {
  192.168.0.0/16;
  172.16.0.0/12;
}
```

Prefix lists:

```
source-prefix-list {
prefix-list1;
prefix-list2;
}
```

NOTE Prefix lists and multiple prefix listings under a term are not order dependent. Junos evaluates the prefixes by longest match. This is the same behavior as with Junos policy.

Noncontiguous Prefix Matching

Also known as *Discontiguous Subnet Masks* in other vendors' operating systems, the *Noncontiguous Prefix* references multiple prefixes that are not adjacent to each other based on normal supernetting rules. For example, the prefix 192.168.0.0/23 is comprised of two /24's: 192.168.0.0/24 and 192.168.1.0/24. To steal a concept from a classic CCIE question: What if you need to match only the odd subnets in a given prefix range? The answer lies in the use of noncontiguous prefix matching. The example shown below can be used to match odd prefixes only:

```
destination-address 192.168.1.0/0.0.254.0;
```

Excluding Prefixes

Prefixes can be excluded from the match conditions under a term. There may be situations where a supernet should be referenced and specific networks should be excluded from the configured action. This is accomplished by specifying the `except` keyword after the prefix:

```
destination-address 192.168.0.0/16;
destination-address 192.168.100.0/24 except;
```

All packets that are destined to any address in the 192.168.0.0/16 range will be subject to actions of the "then" statement, except for those packets destined for 192.168.100.0/24. The 192.168.100.0/24 exception allows the packets to bypass the current term and be processed later in the firewall filter.

Bit-field Matching

Outside of the source and destination address fields in a packet are other optional bit fields that can be useful for packet filtering. These bit-fields contain information about the state of a TCP packet or IP options. All bit-field options must be enclosed in quotations, such as:

```
tcp-flags "rst";
```

IPv4 Protocol Matching

Table 4.2 lists common protocol match conditions in the IPv4 protocol for numeric matches, prefix, and bit-fields.

Table 4.2 Common Protocol Match Conditions

Numeric Match Conditions	Numeric Match Description
destination-port number	Matches a TCP or User Datagram Protocol (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 TCP or protocol UDP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a text synonym. For example, you can specify telnet or 23.
esp-spi spi-value	Matches an IPSec encapsulating security payload (ESP) security parameter index (SPI) value. Match on this specific SPI value. You can specify the ESP SPI value in hexadecimal, binary, or decimal form.
forwarding-class class	Matches a forwarding class. Specify assured-forwarding, best-effort, expedited-forwarding, or network-control.
fragment-offset number	Matches the fragment offset field.
icmp-code number	Matches the ICMP code field. Normally, you specify this match condition in conjunction with the protocol ICMP match statement to determine which protocol is being used on the port. This value or keyword provides more specific information than ICMP-type. Because the value's meaning depends on the associated ICMP-type, you must specify ICMP-type along with ICMP-code. In place of the numeric value, you can specify a text synonym. For example, you can specify ip-header-bad or 0.
icmp-type number	Matches the ICMP packet type field. Normally, you specify this match condition in conjunction with the protocol ICMP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a Matches the ICMP code field. Normally, you specify this match condition in conjunction with the protocol ICMP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a text synonym. For example, you can specify time-exceeded or 11.
interface-group group-number	Matches the interface group on which the packet was received. An interface group is a set of one or more logical interfaces. For information about configuration interface groups, see the *Junos Policy Framework Configuration Guide*, at http://www.juniper.net/techpubs/.

packet-length bytes	Matches the 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	Matches a 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 condition in conjunction with the protocol TCP or protocol UDP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a text synonym. For example, you can specify BGP or 179.
precedence ip-precedence-field	Matches the IP precedence field. You can specify precedence in hexadecimal, binary, or decimal form. In place of the numeric value, you can specify a text synonym. For example, you can specify immediate or 0x40.
protocol number	Matches the IP protocol field. In place of the numeric value, you can specify a text synonym. For example, you can specify OSPF or 89.
source-port number	Matches the 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 condition in conjunction with the protocol TCP or protocol UDP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a text synonym. For example, you can specify http or 80.

Prefix Matching	Prefix Match Description
address prefix	Matches the IP source or destination address field. You cannot specify both the address and the destination-address or source-address match conditions in the same term.
destination-address prefix	Matches the IP destination address field. You cannot specify the destination-address and address match conditions in the same term.
destination-prefix-list prefix-list	Matches the IP destination prefix list field. You cannot specify the destination-prefix-list and prefix-list match conditions in the same term.
prefix-list prefix-list	Matches the IP source or destination prefix list field. You cannot specify both the prefix-list and the destination-prefix-list or source-prefix-list match conditions in the same term.
source-address prefix	Matches the IP source address field. You cannot specify the source-address and address match conditions in the same rule.

source-prefix-list prefix-list	Matches the IP source prefix list field. You cannot specify the source-prefix-list and prefix-list match conditions in the same term.

Bit-field Match	Bit-field Match Description
fragment-flags number	Matches an IP fragmentation flag. In place of the numeric value, you can specify a text synonym. For example, you can specify more-fragments or 0x2000.
ip-options number	Matches an IP option. In place of the numeric value, you can specify a text synonym. For example, you can specify record-route or 7.
tcp-flags number	Matches a TCP flag. Normally, you specify this match condition in conjunction with the protocol TCP match statement to determine which protocol is being used on the port. In place of the numeric value, you can specify a text synonym. For example, you can specify syn or 0x02.
first-fragment	Matches the first fragment of a fragmented packet. This condition does not match unfragmented packets.
is-fragment	Matches the trailing fragment of a fragmented packet. It does not match the first fragment of a fragmented packet. To match both first and trailing fragments, you can use two terms, or you can use fragment-offset 0-8191.
tcp-established	Matches a TCP packet other than the first packet of a connection. This match condition is a synonym for "(ack | rst)". This condition does not implicitly check that the protocol is TCP. To do so, specify the protocol TCP match condition.
tcp-initial	Matches the first TCP packet of a connection. This match condition is a synonym for "(syn & !ack)". This condition does not implicitly check that the protocol is TCP. To do so, specify the protocol TCP match condition.

Policers

When *rate limiting* is required as a part of a packet filter, then *policers* are configured. Policers are the ingress gatekeepers to manage the rate at which transit traffic is allowed to flow through a given interface. Policers limit a given traffic flow to a specified rate with a specified burst rate. For example, if Internet video streaming is adversely affecting the network WAN links, you many want to create a policer that limits that traffic to 128kbps. Policers are configured independent of the firewall filter and may be referenced as an action in one or more firewall filters.

The basic policer syntax is shown here:

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

Based on the configured `if-exceeding` traffic rates, then the following actions can take place:

- discard – drop the traffic exceeding the specified rate
- loss-priority level – change the loss priority
- forwarding-class class-name – change the class-of-service

More detailed explanations of policers are discussed in Chapter 5.

Actions

Just as the "from" statement in a term determines the match conditions, the "then" statement assigns actions to the term. These actions fall into three primary categories.

- terminating actions
- nonterminating actions
- policing actions

Only a single terminating and policing action can be used for a given term, but any number of non-terminating actions may be applied.

Terminating Actions

Terminating actions stop the processing of packets through the firewall filter. There are six terminating actions available for firewall filters:

- accept – accept the packet.
- discard – silently drop the packet.
- reject – drop the packet and send an ICMP-unreachable. You may also specify an ICMP message type after the reject keyword.

- logical system logical-system-name – accepts and forwards packets to a specified logical system.
- routing-instance routing-instance-name – accepts and forwards packets to a specified routing-instance.
- topology topology-name – used with multitopology routing. Accepts and forwards packets to a specific routing topology.

Nonterminating Actions

Nonterminating actions are used to modify packet header information or perform packet accounting functions. Packet header modifications are those actions that change layer 3 or layer 4 attributes, like changing the class of service code point. Accounting can also be performed on the packets, such as J-flow accounting, logging packet information to syslog, or counting the number of matches for a given term.

Table 4.3 Nonterminating Actions

Nonterminating Action	Description
count counter-name	Count the packet in the specified counter.
dscp value	(Family inet only) Classify the packet into one of the following forwarding classes: as, assured-forwarding, best-effort, expedited-forwarding, or network-control. count counter-name Count the packet in the specified counter. DSCP value (Family inet only) Set the IPv4 Differentiated Services code point (DSCP) bit. You can specify a numerical value from 0 t°hrough 63. To specify the value in hexadecimal form, include 0x as a prefix. To specify the value in binary form, include b as a prefix. NOTE: The actions *DSCP 0* or *DSCP be* (best effort) are supported only on TSeries and M320 routers and on the10-Gigabit Ethernet Modular Port Concentrators (MPC), 60-Gigabit Ethernet MPC, 60-Gigabit Ethernet Queuing MPC, and 60-Gigabit Ethernet Enhanced Queuing MPC on MX Series routers. However, these actions are not supported on Enhanced III Flexible PIC Concentrators (FPCs) on M320 routers.
forwarding–class class	Classify the packet into a user-defined forwarding class or one of the following default forwarding classes: assured-forwarding, best-effort, expedited-forwarding, or network-control.

log	(Family inet and inet6 only) Log the packet header information in a buffer within the Packet Forwarding Engine. You can access this information by issuing the show firewall log command at the command-line interface (CLI).
loss-priority (high | medium-high | medium-low| low)	Set the loss priority level for packets. Supported on MX Series routers; M120 and M320 routers; and M7i and M10i routers with the Enhanced CFEB (CFEB-E). On M320 routers, you must enable the tricolor statement at the [edit class-of-service] hierarchy level to commit a PLP configuration with any of the four levels specified. If the tricolor statement is not referenced, you can only configure the high and low levels. This applies to all protocol families. You cannot also configure the three-color-policer nonterminating action for the same firewall filter term. These two nonterminating actions are mutually exclusive.
next term	Continue to the next term for evaluation.
policer policer-name	Using the specified policer, rate-limit the packets.
port-mirrror	(Family bridge, ccc, inet, inet6, and vpls only) Port-mirror packets based on the specified family. Supported on M120 routers, M320 routers configured with Enhanced III FPCs, and MX Series routers only.
sample	(Family inet, inet6, and mpls only) Sample the packets.using J-flow accounting.
syslog	Log the packet to the system log file.
three-color-policer policer-name	Apply rate limits to the traffic using the tricolor marking policer. You cannot also configure the loss-priority action modifier for the same firewall filter term. These two action modifiers are mutually exclusive.
traffic-class value	(Family inet6 only) Specify the traffic-class code point. You can specify a numerical value from 0 through 63. To specify the value in hexadecimal form, include 0x as a prefix. To specify the value in binary form, include b as a prefix. The default traffic-class value is best effort, that is, be or 0. Note: The actions traffic-class 0 or traffic class be (best effort) are supported only on TSeries and M320 routers and on the 10-Gigabit Ethernet Modular Port Concentrator (MPC), 60-Gigabit Ethernet MPC, 60-Gigabit Ethernet Queuing MPC, and 60-Gigabit Ethernet Enhanced Queuing MPC on MX Series routers. However, these actions are not supported on Enhanced III Flexible PIC Concentrators (FPCs) on M320 routers.

Nonterminating actions carry the implicit terminating action of *accept*. When applied to a firewall filter term without an explicit terminating action, the default action of accept will be used. This could cause unintended packet processing side effects if you are just looking to sample or log a packet. To avoid the implicit accept action, use the next term action to allow further processing of the packets within the firewall filter.

NOTE The next term action is not compatible with terminating actions. Either the next term statement is used or a terminating action is configured, or else the implicit accept is used.

Unlike terminating actions, one or more nonterminating actions may be combined as an action in addition to the terminating action. The next example shows a single firewall-filter `example-filter` that is sysloging and counting all packets from hosts on the 192.168.2000/24 network that are connecting to web servers:

```
 [edit firewall family inet]
jack@west# show
filter example-filter {
   term web-srvr {
      from {
         source-address {
            192.168.200.0/24;
         }
         destination-port http;
      }
      then {
         count websrvr-count;
         syslog;
         accept;
      }
   }
   term last {
      then accept;
   }
}
```

The example shows multiple non-terminating actions with the single terminating action of accept.

BEST PRACTICE Always define a terminating action or use `next term`. There are too many implicit rules, default actions, and different vendor implementations to "guess" what a policy or packet filter may do. Plus, your other team members might not share your expert knowledge. Be safe – Be explicit!

Firewall Evaluation Logic

So far, this chapter has alluded to the fact that firewall filters are processed in a top down fashion. The filter itself is a container that holds multiple terms. The terms contain the match conditions and actions that determine what will happen to the packet as it is evaluated.

It's important to note that firewall filters are stateless and subsequently only evaluate packets in a unidirectional manner. This means that blocking the packet flow from a source to a destination does not infer the reverse action of blocking traffic from a destination back to the source. Generally speaking, this is effective in preventing two-way communication between the two hosts. Don't assume that this secures either host, however. Many attacks use techniques to deny access to hosts, which can be realized in a unidirectional fashion by flooding or resource depletion.

Let's revisit a graphic from Chapter 1.

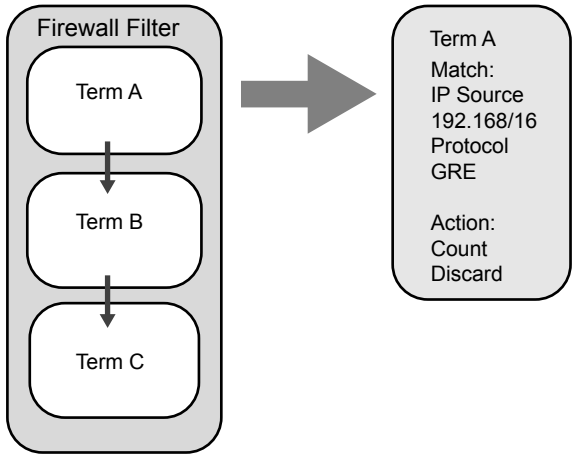


Figure 4.1 Firewall Filter Packet Processing Flow

Figure 4.1 shows a visual representation of how a firewall filter evaluates a packet. Using two different packets, let's walk through the process but use a best guess since only Term A is shown.

- GRE packet sourced from 192.168.1.10
- HTTP packet sourced from 172.16.177.10

The GRE packet is evaluated by the firewall filter and begins with the

first term, Term A. Because it matches the defined source IP address and the correct protocol, the packet is counted and discarded. The HTTP does not match the first term, so it is processed by the subsequent terms B and C.

Let's explore a more real world example:

```
[edit firewall family inet filter example-filter]
jack@west# show
term a {
    from {
        source-address {
            192.168.1.0/24;
        }
        destination-address {
            192.168.3.0/24;
        }
    }
    then {
        log;
        accept;
    }
}
term b {
    from {
        source-address {
            192.168.10.0/24;
        }
        destination-address {
            192.168.30.0/24;
        }
        source-port [ 5004 5060 10000 16348-32768 ];
    }
    then {
        forwarding-class expedited-forwarding;
        accept;
    }
}
term c {
    from {
        source-address {
            192.168.10.10/32;
        }
        destination-address {
            192.168.30.10/32;
        }
    }
    then {
        discard;
    }
}
```

This firewall-filter, `example-filter`, is applied in an inbound direction on the interface shown in drawing Figure 4.2.

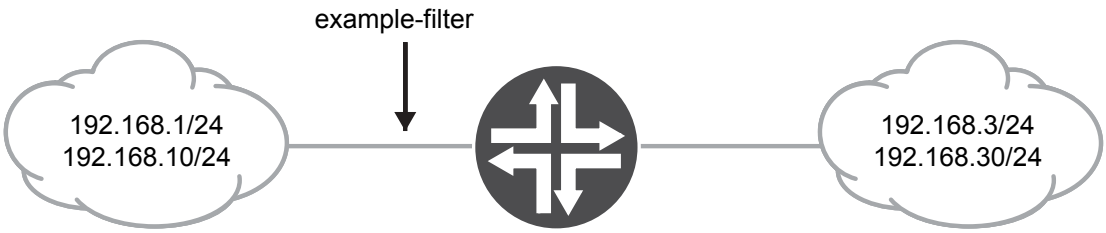


Figure 4.2 Example-Filter Is Applied in the Inbound Direction

Using the same term nomenclature – terms A, B, and C – let's evaluate the following prefixes:

- Generic packets from 192.168.1.10 destined for 192.168.3.10
- Traffic between 192.168.10.10 and 192.168.30.10
- SIP packet sourced from 192.168.10.10 destined for any host on 192.168.30.0/24
- HTTP packets from 192.168.10.100 destined for 192.168.30.199

The first test set, generic packets sourced by 192.168.1.10 and destined for 192.168.3.10, is evaluated against the packet filter `example-filter`. Based on the match conditions of term A, it is determined that this flow is an exact match and the packets are logged and accepted by the firewall filter. All further terms are not evaluated.

The next test set needs to be evaluated against the example packet filter. Traffic between the two hosts 192.168.10.10 and 192.168.30.10 does not match the criteria in either term A or term B. Term C is a match. Since the only qualifying match conditions are source and destination address, this packet stream will be silently dropped.

The third test set is concerned with SIP traffic flowing between the host 192.168.10.10 and the destination subnet of 192.168.30.0/24. Evaluating from top to bottom, term A does not match, but term B criteria matches SIP traffic. Traffic matching term B is moved to the forwarding-class expedited forwarding, since it is VoIP traffic, and accepted.

The final test set is web traffic flowing from 192.168.10.100 to

192.168.30.199. Evaluating this traffic flow against `example-filter`, it is noticed that the traffic does not match any of the terms. So what action will be applied to this test set?

NOTE The default action for a term is to accept packets that match the criteria defined in the "from" statement. However, this traffic flow does not match any of the terms so the default filter behavior is invoked, which is an implicit deny-all.

The traffic will be silently dropped.

Firewall Filter Chains

There is an implicit firewall filter that is appended to the end of all user configured packet filters. This implicit filter drops all packets. Without an explicit `accept` in user-defined policy, every packet that does not match a term that has a corresponding action of accept will be discarded.

This illustrates an interesting concept of chaining firewall filters together. If, by default, an implicit `deny all` filter is appended to your user-defined policies, then it would be possible to link user defined filters into a chain. Junos interprets firewall filter chains as a single firewall filter. Since firewall filters are compiled into the forwarding table, the packet filters are simply merged together and the terms are processed sequentially in the order that they appear in overall filter chain.

```
[edit]
jack@west# show interfaces fe-0/0/6
unit 0 {
    family inet {
        filter {
            input-list [ example-filter example2 ];
        }
        address 192.168.12.2/24;
    }
}
```

Here you can see the original filter example, `example-filter`, applied to the interface fe-0/0/6. Added to the filter list is a new firewall filter `example2`:

```
[edit firewall family inet filter example2]
jack@west# show
term d {
    from {
```

```
        source-address {
            192.168.10.100/32;
        }
        destination-address {
            192.168.30.0/24;
        }
        protocol tcp;
        port [ http https ];
    }
    then accept;
}
```

The effect of this change is that HTTP traffic and HTTPS traffic are now allowed, along with packets that match the original firewall filter. This also satisfies the final criteria from the test set that we evaluated against the first firewall filter. When the configuration is committed in Junos, the merged firewall filters become a superset of both individual filters.

Summary

How are firewall filter chains helpful in the real world? They give you the ability to create standard packet filters that address universal security concerns so that individual filters do not have to be created for every individual interface in the network. Write the filter once and apply it universally to all devices. For interfaces that require additional and more fine grained filters, simply create the interface-specific filter and apply both filters using the `input-list` or `output-list` syntax under the appropriate interface.

Note that packet filters shouldn't be applied to *every* single router interface in the network. Filters should be applied where packet filtering makes sense.

Chapter 5

Policer Configuration

Policers are an important mechanism to rate-limit and to generally affect how transit traffic is handled in the network. From the straightforward to the hierarchical, this chapter breaks down policers into their fundamental components. Policers are important components for use with class of service and firewall filters. Junos supports several policing methods.

Policer Types

Junos supports three types of policers. While the descriptions seem daunting, you'll realize that these policer types are the same rate-limiters you have been using all along.

- Single-rate two-color policer
- Single-rate three-color policer
- Two-rate three-color policer

What's up with the colors? Think of a traffic signal: green means *go*, yellow means *caution*, and red means *stop*.

When traffic is conforming to the specified policer rate, the traffic is allowed to flow normally. Traffic that is above the configured rate and burst, but has not exceeded the excess rate, is in the caution zone. Once traffic exceeds the configured upper threshold of a policer, then it is in the red zone and is discarded. The color designations are for visualization of the policer behavior – you don't actually configure colors.

Single-rate Two-color Policer

The *single-rate two-color* policer is the most common policer used in networks today. Simply stated, traffic that is within contract, or specified bandwidth and burst rate, is not affected by the policer. Traffic that exceeds the configured contract rate can be marked with a higher loss priority, placed into a different forwarding class, or discarded.

Single-rate means that there is only a single bandwidth and burst rate referenced in the policer. The two colors associated with this policer are green and red.

Color	Implicit Action	Configurable Action
Green (Conforming)	Assign Low Loss Priority	None

Red (Nonconforming)	None	Assign low or high loss priority, assign a forwarding class, or discard. On some platforms, you can assign medium-low or medium-high loss priority.

Here is a sample single-rate two color policer:

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

CAUTION You may choose either `bandwidth-limit` or `bandwidth-percent`, as they are mutually exclusive. You cannot configure a policer to use bandwidth percentage for aggregate, tunnel, and software interfaces.

The single-rate two-color policer is the workhorse for most network configurations. It is used with packet filters, multifield classifiers for class of service, and interface rate limiting. Being easy to configure and extremely flexible makes this the "go to" policer type.

Burst Size

Determining the burst-size for a policer is usually a point for debate. The recommended formula for calculating burst size for bandwidth described as bits per second is:

```
burst size = bandwidth x allowable time for burst traffic / 8
```

For policers where the interface bandwidth is unknown, use the MTU method of calculating burst size:

```
Burst size = Interface MTU x 10
```

NOTE There is finite buffer space for an interface. A good rule of thumb estimate of the total buffer depth for an interface is around 125ms. When configuring burst size keep this in mind.

Single-Rate Three-Color Policer

The *single-rate three-color* policer is similar to the single-rate two-color policer with the addition of the yellow color. The single-rate two-color policer addresses conforming and nonconforming traffic. Single-rate three-color policers introduce the idea of a *committed information rate* (CIR) as well as a *committed burst rate* (CBR). Traffic rates below the CIR are conforming. Traffic below the CIR and CBR is conforming and no action is taken. Traffic that reaches the excess burst size (EBS) is discarded. Traffic that is above the CIR and CBR but below the EBS is assigned a higher-loss priority, making it more susceptible to being dropped during congestion. This concept is very similar to the frame-relay discard-eligible bit.

Color	Implicit Action (internal to router)	Configurable Action
Green (Conforming)	Assign Low Loss Priority	None
Yellow (Exceeds CIR and CBR)	Assign Medium-high Loss Priority	None
Red (exceeds EBS)	Assign High Loss Priority	Discard

And here is a sample single-rate three-color policer:

```
[edit firewall]
three-color-policer name {
  action {
    loss-priority high then discard;
  }
  logical-interface-policer;
  single-rate {
    (color-aware | color-blind);
    committed-information-rate bps;
    committed-burst-size bytes;
    excess-burst-size bytes;
  }
}
```

MORE? As defined by RFC 2697, *A Single Rate Three Color Marker*, this policer actually adjusts the loss priority in the DSCP field of the packet.

NOTE Three-color-policers can be configured as color-aware or color-blind in the Junos OS. If the policer is color-aware then the loss priority can

only be marked higher – even if the packet is conforming to the policer as it transits the router. In color-blind mode, Junos ignores the existing loss priority on the packet and marks the loss priority, higher or lower, based on the policer's implicit action.

Two-Rate Three-Color Policer

The *two-rate three-color* policer improves on the single-rate three-color policer by introducing a second rate tier. Reviewing the single-rate three-color policer, there is only an excess burst size above the committed rate and burst size. Two-rate three-color policers expand the second tier to include both an upper bandwidth limit and associated burst size, peak information rate (PIR), and a peak burst size (PBS).

Color	Implicit Action (internal to router)	Configurable Action
Green (Conforming)	Assign Low Loss Priority	None
Yellow (Exceeds CIR and CBR)	Assign Medium-high Loss Priority	None
Red (exceeds PIR and PBS)	Assign High Loss Priority	Discard

MORE? Two-rate three-color policers are defined by RFC 2698, *A Two Rate Three Color Marker*.

Here is a sample two-rate three-color policer:

```
[edit firewall]
three-color-policer name {
  action {
    loss-priority high then discard;
  }
  logical-interface-policer;
  two-rate {
    (color-aware | color-blind);
     committed-information-rate bps;
     committed-burst-size bytes;
     peak-information-rate bps;
     peak-burst-size bytes;
  }
}
```

Miscellaneous Policer Information

There are some miscellaneous options to be aware of when configuring policers and the following sections detail these instances.

Order of Operations: Policers and Firewall Filters

There is an inherent order to the operations of all computing devices. Junos is no different. Figure 5.1 represents the order in which policers and firewall filters are referenced by Junos.

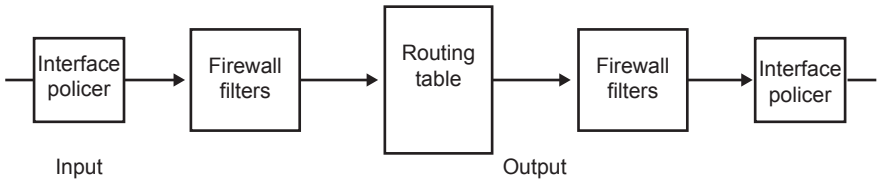


Figure 5.1 Order of Policers and Filters by Junos

For policers and firewall filters that are applied on ingress, the policer takes precedence before the firewall filter is evaluated. This does not include policers referenced within a firewall filter – only policers that have been applied directly to the interface. Inversely, firewall filters are processed before the interface policers when applied in the egress direction.

Multiple policers can be applied and evaluated for a given ingress interface. Queue level policers are evaluated before policers applied at the logical interface level. MAC layer policers (Layer 2) are evaluated last. For egress policing, only a single policer may be configured.

Policer Configuration Options

There are particular keywords that may be used when configuring policers that affect the way the policer is handled by Junos:

logical-bandwidth-policer: Configuring the policer bandwidth-percent uses the physical interface bandwidth associated with the actual media type. If a shaper is applied to the interface, the logical-bandwidth-policer will enable the policer to reference the shaped rate.

logical-interface-policer: Each application of a policer enables a separate instance of the policer. The logical-interface-policer keyword creates an aggregate instance in which all applications of the policer are treated as an aggregate for a given logical interface.

physical-interface-policer: The physical-interface-policer aggregates the bandwidth constraints for all logical interfaces belonging to the same physical interface. This keyword works across multiple routing instances.

filter-specific: Policers operate as independent entities when referenced in a firewall filter term. The filter-specific keyword aggregates the behavior of the policer at the firewall filter level.

Policers and Firewall Filters

Most policers are combined with a firewall filter to selectively rate-limit traffic based on the match conditions specified by the firewall filter. Policers by themselves do not have a mechanism to differentiate between different types of traffic. This combination makes a powerful tool to manage traffic flows.

Policers are applied as an action for a given term within a firewall filter. The policer is applied along with the other nonterminating actions and is subject to the terminating action. Only a single policer statement can be applied per term.

The following example configuration shows a policer that limits best effort traffic to 1Mbps, and the policer will discard traffic exceeding 1Mbps:

```
[edit]
firewall {
   policer 1m-policer {
      if-exceeding {
         bandwidth-limit 1m;
         burst-size-limit 125k;
      }
      then discard;
   }
   family inet {
      filter police-some-traffic {
         term 1 {
            from {
               dscp be;
            }
            then {
               policer 1m-policer;
               accept;
            }
         }
         term default {
            then accept;
         }
      }
   }
}
```

Per-Prefix Specific Actions

Up until this point, you have seen policers at the network and interface levels, but Junos also provides more control for dealing with policing at the *per-prefix* level. Whether you want to apply a policer for every /32 in a given subnet, or to create a single policer for every /24 in a /16, per-prefix policers are the tools you use to complete the job.

There are few things to remember about per-prefix actions:

- Per-prefix policers generate multiple policers when complied into the forwarding plane. Remember that routers have finite resources so don't configure every policer in your network as a per-prefix policer.
- Per-prefix policers are only configurable for the IPv4 protocol family.
- Per-prefix policers are not supported on SRX and J-series devices.

To configure per-prefix policers use the following syntax.

```
[edit firewall family inet]
prefix-action name {
   count;
   destination-prefix-length prefix-length;
   policer policer-name;
   source-prefix-length prefix-length;
   subnet-prefix-length prefix-length;
}
```

The `subnet-prefix-length` is what sets the top prefix-length index. The source-prefix-length and/or destination-prefix-length set the low side of the repeating pattern. So, for each source or destination defined prefix for a given subnet, generate a unique policer. The number of policers generated is determined by the following formula:

```
Number = 2 ^ (source/destination-prefix-length - subnet-prefix-length)
```

To set a prefix action for all /32 hosts for a given /24 see this example:

```
[edit]
firewall {
          policer host-policer {
            filter-specific;
            if-exceeding {
              bandwidth-limit 1m;
              burst-size-limit 128k;
            }
            then {
              discard;
```

```
            }
          }
          family inet {
            prefix-action prefix-policer-set {
              count;
              destination-prefix-length 32;
              policer host-policer;
              subnet-prefix-length 24;
            }
          }
          filter filter-hosts {
            term term1 {
              from {
                destination-address 192.168.100/24;
              }
              then {
                prefix-action prefix-policer-set;
              }
            }
          }
        }
}
```

Notice that the `subnet-prefix-length` in the prefix-action matches the destination prefix-length in the firewall filter. This prevents the generated policers from overlapping and the Junos OS will create 256 1Mbps policers in this example.

Adding additional destination addresses to the firewall filter will cause the reuse of some of the 256 policers.

What if you wanted to apply a 50 Mbps policer per subnet for every /24 network in the RFC1918 address 172.16/16?

Well, review the following:

```
[edit]
firewall {
        policer network-policer {
          filter-specific;
          if-exceeding {
            bandwidth-limit 50m;
            burst-size-limit 256k;
          }
          then {
            discard;
            }
          }

          family inet {
            prefix-action prefix-policer-set {
            count;
```

```
            destination-prefix-length 24;
            policer network-policer;
            subnet-prefix-length 16;
          }
          filter limit-networks {
            term term1 {
              from {
                destination-address 172.16.0.0/16;
              }
              then {
                prefix-action prefix-policer-set;
              }
            }
          }
        }
}
```

The prefix action subnet prefix length matches the filter `limit-networks` destination address prefix length (both are /16). The prefix action generates 256 unique policers based on the configuration. For every host contained in each unique /24 network – 172.16.0.0/24, 172.16.1.0/24, 172.16.2.0/24 ... 172.16.255.0/24 – each will be governed by a common policer.

Interface Policers

Policers are only useful when they are combined with another part of the configuration. The initial act of policer configuration creates a policer template. This section breaks down three common interface policer configurations.

Policers are configured under an interface for a particular protocol family. The sample code below shows the syntax required to apply a policer to an interface:

```
[edit interfaces]
ge-0/0/0 {
      unit 0 {
        family inet {
           policer {
              input policer-name;
              output policer-name;
           }
      }
   }
 }
```

NOTE Policers and firewall filters can coexist on an interface. It is imperative to remember the order of operations illustrated in Figure 5.1.

Physical Interface Policers

Physical interface policers are used to aggregate and limit the total available bandwidth across multiple logical interfaces as well as multiple protocol family instances.

To create the policer:

```
[edit firewall]
jack# show
policer phy-int-policer {
    physical-interface-policer;
    if-exceeding {
        bandwidth-limit 50m;
        burst-size-limit 256k;
    }
    then discard;
}
```

Then apply the policer to an interface:

```
[edit]
jack# show interfaces fe-0/0/7
unit 0 {
    family inet {
        policer {
            input phy-int-policer;
        }
        address 192.168.12.2/24;
    }
}
```

The policer may also be referenced and applied with a firewall filter. Here's an alternate configuration:

```
[edit firewall family inet filter match-and-police]
jack@west# show
physical-interface-filter
term 1 {
    from {
        source-address {
            192.168.100.10/24;
        }
    }
    then policer phy-int-policer;
}
term last {
    then accept;
}
```

And the firewall filter containing the physical interface policer is then applied to the interface:

```
[edit]
jack@west# show interfaces fe-0/0/7
unit 0 {
    family inet {
        filter {
            input match-and-police;
        }
        address 192.168.12.2/24;
    }
}
```

There are some caveats to remember here:

- Both physical interface-policing methods are mutually exclusive. You may apply the policer directly to the interface or use a firewall filter, but not both.
- Physical interface policers are not available on the SRX or J-series devices.
- You cannot create a policer that contains the `physical-interface-policer` and `interface-specific` keywords.
- Firewall filters must be configured under a specific protocol family. Family `any` is not supported.

Aggregate Policers

If you need to rate-limit traffic across several different protocol families from the same interface, forcing them to share to the same bandwidth constraints, you use an *aggregate policer*. Imagine a customer-facing interface is configured for both IPv4 and IPv6 and you want to police the traffic to 50 Mbps for that interface, regardless of the protocol being used. For this an aggregate policer is configured using the `logical-interface-policer` keyword:

```
 [edit firewall]
jack# show
policer log-int-policer {
    logical-interface-policer;
    if-exceeding {
        bandwidth-limit 50m;
        burst-size-limit 256k;
    }
    then discard;
}
```

After configuring the policer, you apply the same policer to all configured protocol families for a given interface:

```
[edit]
jack@west# show interfaces fe-0/0/7
unit 0 {
    family inet {
     policer {
           input log-int-policer;
        }
        address 192.168.12.2/24;
    }
    family inet6 {
        policer {
           input log-int-policer;
        }
        address ff80:1000::1/64;
    }
}
```

This ensures that all traffic, both IPv4 and IPv6, is rate-limited under a single 50 Mbps cap.

Bandwidth Policers

An alternate configuration to limiting bandwidth by a precise rate is to use a more ambiguous bandwidth percentage. By default, the bandwidth is determined by the physical port speed. When a *shaper* is applied under the `class-of-service` stanza, however, the bandwidth percentage will use the shaped rate as the base interface bandwidth. Here is an example of a shaper applied to a fast-ethernet interface:

```
[edit]
jack# show class-of-service
interfaces {
    fe-0/0/7 {
        unit 0 {
           shaping-rate 50m;
        }
    }
}
```

If a subsequent policer was added to the same interface, the bandwidth percentage would no longer be 10% of 100Mbps, instead it would be 10% of the shaped rate of 50Mbps, as such:

```
[edit firewall]
jack# show
policer band-percent-policer {
   logical-bandwidth-policer;
   if-exceeding {
      bandwidth-percent 10;
      burst-size-limit 128k;
   }
   then discard;
}
```

Summary

Policing and shaping are important tools to control traffic and keep it in conformance. This chapter covered the various methods of policing and their applications in the Junos configuration. Three supported policing types are:

- Single-rate two-color policers
- Single-rate three-color policers
- Two-rate three-color policers

Policers are useful for rate-limiting traffic, while shaping is useful for normalizing traffic flows on a given interface. When used together with CoS, these features provide a way to manage the traffic flows through to the router to ensure delivery of critical traffic.

Applying physical and logical interfaces provides an additional level of traffic grooming. The configurations illustrated in this chapter should help you configure policers and shapers on your own network.

MORE? If you want to copy and paste the configurations and policies used in this book, check out the *Copy and Paste* edition of this book at http://www.juniper.net/dayone.

What to Do Next & Where to Go ...

http://www.juniper.net/dayone

The *Day One* book series is available for free download in PDF format. Select titles also feature a *Copy and Paste* edition for direct placement of Junos configurations. The library is available in eBook format for iPads and iPhones from the iTunes Store>Books, or download to Kindles, Androids, Blackberrys, Macs, and PCs by visiting the Kindle Store. In addition, print copies are available for sale at Amazon or www.vervante.com.

http://www.juniper.net/books

Juniper Networks Books works with reputable book publishers around the world to publish networking books for use in the field or classroom that are authored, edited, or reviewed by Juniper Networks subject matter experts and engineers. Check out the complete Juniper Networks Books library for new releases every calendar quarter.

http://forums.juniper.net/jnet

The Juniper-sponsored J-Net Communities forum is dedicated to sharing information, best practices, and questions about Juniper products, technologies, and solutions. Register to participate in this free forum.

www.juniper.net/techpubs/

Juniper Networks technical documentation includes everything you need to understand and configure all aspects of Junos, including MPLS. The documentation set is both comprehensive and thoroughly reviewed by Juniper engineering.

www.juniper.net/training/fasttrack

Take courses online, on location, or at one of the partner training centers around the world. The Juniper Network Technical Certification Program (JNTCP) allows you to earn certifications by demonstrating competence in configuration and troubleshooting of Juniper products. If you want the fast track to earning your certifications in enterprise routing, switching, or security use the available online courses, student guides, and lab guides.