

DAY ONE: ADVANCED IPV6 CONFIGURATION

It's day one and you need to enable BGP routing in your network. Where to start? Start here and get it done with Junos. You'll configure, test, and verify your IPv6 BGP configurations with best practices and rock-solid techniques.

By Chris Grundemann

DAY ONE: ADVANCED IPV6 CONFIGURATION

Day One: Advanced IPv6 Configuration is the second book in the Junos® Networking Technologies Series on IPv6. The first book, *Day One: Exploring IPv6*, introduced all the basics of configuring an IPv6 enabled LAN: interface addressing, static routes, neighbor discovery, and IGP routing. Now you're ready to complete the configuration and testing tasks required to enable BGP routing in your network. You'll learn how to set up both Internal Border Gateway Protocol (IBGP) and External Border Gateway Protocol (EBGP) with IPv6, and how to leverage native IPv6 peering. You'll also learn how to test and verify your IPv6 BGP configurations. So roll up your sleeves and let's get to work.

"This book is a fantastic tutorial on configuring and testing BGP routing with IPv6 on your network. It's completely hands-on. It also covers native IPv6 peering and how to advertise IPv6 routes over IPv4 peering sessions. Highly recommended."

Owen DeLong, IPv6 Evangelist, Hurricane Electric

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

- Configure BGP for IPv6, including IBGP and EBGP in Junos.
- Understand the use of the IPv6 NLRI in MP-BGP.
- Verify the proper operation of IPv6 BGP peering.
- Use VRRP for IPv6 to add redundancy and quicker failover.
- Implement CoS on an IPv6 network.
- Explain the basics of Multicast Listener Discovery (MLD).
- Understand the wide variety of options available for systems management in IPv6.
- Set up a production IPv6 network based on the success of your testbed and the results and feedback that testbed provides.

Juniper Networks Day One books provide just the information you need to know on day one. That's because they are written by subject matter experts who specialize in getting networks up and running. Visit www.juniper.net/dayone to peruse the complete library.

Published by Juniper Networks Books

ISBN 978-193677920-8



5 1400



JUNIPER
NETWORKS®

Junos® Networking Technologies Series

Day One: Advance IPv6 Configuration

By Chris Grundemann

<i>Chapter 1: Exploring BGP Support for IPv6</i>	<i>5</i>
<i>Chapter 2: Getting Ready for Production IPv6.....</i>	<i>29</i>
<i>Chapter 3: Discovering IPv6 Enabled System Management.....</i>	<i>51</i>
<i>What to Do Next & Where to Go</i>	<i>68</i>

© 2011 by Juniper Networks, Inc. All rights reserved.

Juniper Networks, the Juniper Networks logo, Junos, NetScreen, and ScreenOS are registered trademarks of Juniper Networks, Inc. in the United States and other countries. Junose is a trademark of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice. Products made or sold by Juniper Networks or components thereof might be covered by one or more of the following patents that are owned by or licensed to Juniper Networks: U.S. Patent Nos. 5,473,599, 5,905,725, 5,909,440, 6,192,051, 6,333,650, 6,359,479, 6,406,312, 6,429,706, 6,459,579, 6,493,347, 6,538,518, 6,538,899, 6,552,918, 6,567,902, 6,578,186, and 6,590,785.

Published by Juniper Networks Books

Writer: Chris Grundemann

Editor in Chief: Patrick Ames

Copyediting and Proofing: Nancy Koerbel

Junos Program Manager: Cathy Gadecki

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.

Follow the Day One series on Twitter: @Day1Junos

ISBN: 978-1-936779-20-8 (print)

Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-21-5 (ebook)

Version History: v1 April 2011

2 3 4 5 6 7 8 9 10 #7100139-en

About the Author

Chris Grundemann specializes in the design, implementation, and operation of large IP, Ethernet, and Wireless Ethernet networks and is deeply involved in the policy and politics surrounding internetworking and the Internet. He is JNCIE-M #449 and is currently engaged with tw telecom inc., where he is responsible for leading technology efforts toward the evaluation, design, implementation, and maintenance of existing and next-generation technologies.

Chris is the author of *Day One: Exploring IPv6*. He is the founding Chair of CO ISOC, the Colorado chapter of the Internet Society, and an elected member of the ARIN Advisory Council (AC). Chris is the founding editor of Burning With The Bush, a Juniper Networks focused news and information site, as well as The IPv6 Experts .net, a site dedicated to connecting folks with IPv6 experts and expert information. He also maintains a personal weblog (<http://weblog.chrisgrundemann.com>) aimed towards Internet related posts typically focusing on network operation and design, tech-policy, and the future of the Internet.

Author's Acknowledgments

The author would like to express his deepest gratitude to everyone who contributed to the creation of this book. Patrick Ames was again absolutely crucial as both editor and instructor. Owen DeLong provided much needed technical review and sanity checks. Cathy Gadecki rounded up experts when needed and kept the project rolling. Nancy Koerbel made sure my writing here is legible. Thank you to Becca Nitzin for her careful technical reviews, and to Eddie Parra for the time he spent reviewing the book. The IETF and all of its contributors created the open technical standards that define and drive the IPv6 protocol, without which this book (and possibly the future of the Internet) would not exist. Above all I would like to thank my wife, Erin Grundemann, for giving me the time, support, and love that it takes to do all of the things I do, including writing these books. Thank you all!

What You Need to Know Before Reading this Book

This book assumes you, the reader, are versed in the following concepts:

- ✓ Junos CLI and the Junos operating system.
- ✓ Networking protocols and their usage in medium to large networks, such as BGP, OSPF, and IS-IS
- ✓ Troubleshooting medium to large networks using Junos.
- ✓ The IPv6 protocol and IPv6 addresses.
- ✓ Configuration of basic IPv6 connectivity in Junos, including, at a minimum, interface addressing and static routes.
- ✓ Basic network and system operation using Junos.

This book is written for engineers with experience in either Enterprise or Service Provider networks, and is primarily for those who have some experience working with IPv6 in the Junos operating system. It is written in such a way that it can also serve as a refresher for more experienced users.

After Reading this Book, You'll Be Able to...

- ✓ Configure BGP for IPv6, including IBGP and EBGP in Junos.
- ✓ Understand the use of the IPv6 NLRI in MP-BGP.
- ✓ Verify the proper operation of IPv6 BGP peering.
- ✓ Leverage DHCPv6 to provide more information and more control of dynamic addressing.
- ✓ Use VRRP for IPv6 to add redundancy and quicker failover.
- ✓ Implement CoS on an IPv6 network.
- ✓ Explain the basics of Multicast Listener Discovery (MLD).
- ✓ Understand the wide variety of options available for systems management in IPv6.
- ✓ Set up a production IPv6 network, based on your success in your testbed and the results and feedback that testbed provides.

Day One IPv6 Series

This book makes significant reference to an earlier book in the Day One series, *Day One: Exploring IPv6*.

To obtain the book, you can:

- Get the free PDF edition at www.juniper.net/dayone
- Get the ebook edition for iPhones and iPads using your device's iBook app. Open the iBook, open the iBookstore, search for “Day One Juniper” and download the book.
- Get the ebook edition for Kindle apps by opening your device's Kindle app and going to the Kindle Store. Search for “Day One Juniper.”
- Purchase the paper edition at either Vervante Corporation (www.vervante.com) or Amazon (www.amazon.com).

Dedication: Nathan Day, 1979-2010

This book is dedicated to an absolutely amazing engineer, a wonderful friend, a caring husband, and a loving father. You are missed Nate!

Chris Grundemann

Chapter 1

Exploring BGP Support for IPv6

<i>IPv6 Test Bed</i>	6
<i>Introducing BGP Routing with IPv6</i>	8
<i>Understanding Native IPv6 Peering</i>	9
<i>Advertising IPv6 Routes Over IPv4 Sessions</i>	18

The first book in this Day One series on IPv6, *Exploring IPv6*, introduced all the basics of configuring an IPv6 enabled LAN, interface addressing, static routes, neighbor discovery, and IGP routing. If you haven't read it, you should, in order to obtain step-by-step instructions on configuring IPv6 basics using the Junos operating system. This chapter takes the test bed network used in *Exploring IPv6* to the next level with Border Gateway Protocol (BGP), so a certain familiarity with that test bed network will greatly assist you. *Day One: Exploring IPv6* can be downloaded at www.juniper.net/dayone, and is available in both print and ebook formats. Good reading.

Once you've read *Exploring IPv6*, this book will allow you to complete all of the configuration and testing tasks required to enable BGP routing in your network. In the first chapter you will learn how to set up both Internal Border Gateway Protocol (IBGP) and External Border Gateway Protocol (EBGP) with IPv6, how to leverage native IPv6 peering, and how to advertise IPv6 routes over IPv4 peering sessions. You will also learn how to test and verify your IPv6 BGP configurations. Roll up your sleeves.

ALERT! This isn't your normal alert. The author highly recommends, encourages, and suggests that before taking on the tasks in this chapter, you should have a working familiarity with the Junos CLI, BGP, and basic IPv6 routing in Junos.

MORE? For more on the Junos CLI, basic IPv6 routing in Junos, and other pertinent topics, download *any* of the Day One library titles at www.juniper.net/dayone. The list of titles ranges from elementary to introductory to advanced, so you can find the appropriate title to match your Junos skill level.

IPv6 Test Bed

Okay, enough of telling you this book is about advanced IPv6 configuration; it's time to start showing you. This book picks up right where *Day One: Exploring IPv6* left off, and uses the test bed network we worked so hard creating in that book. All of the examples used here, along with all of the Try It Yourself sections included in this book, are taken directly from the network illustrated in Figure 1.1. It's the same network built in *Exploring IPv6* with the addition of one router, router P1, which allows an EBGp session to be built.

Just as in *Exploring IPv6*, you should follow along in your own lab or other nonoperational environment to get the most out of this book, while diving deeper into IPv6 and the Junos OS.

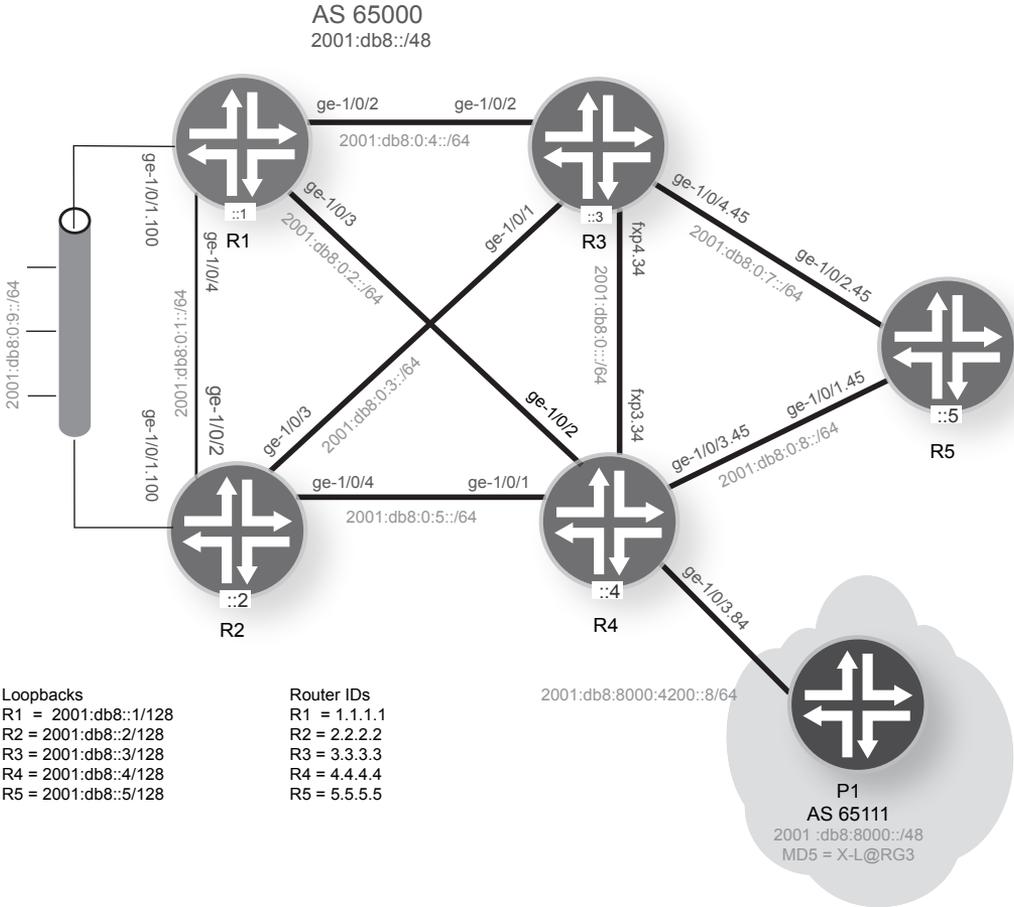


Figure 1.1 IPv6 Test Bed Topology and Addressing

As you examine Figure 1.1, note that the examples and the Try It Yourself sections in this book assume that you have already configured and verified all of the topics covered in *Exploring IPv6* in your test bed network. This book’s test bed uses the OSPF3 configuration from *Exploring IPv6* for its IGP.

Introducing BGP Routing with IPv6

The Border Gateway Protocol (BGP) is the de facto standard for inter-AS routing, and as the exterior gateway protocol (EGP) in most widespread use today, it is the best (and typically the only) way to dynamically connect your autonomous system to other autonomous systems. If you currently use BGP on your IPv4 network, you will most likely need to configure BGP to support IPv6 as well. If you are turning up a new IPv6 network that is multi-homed, or needs dynamic routing updates from another AS, you will surely need to configure BGP for IPv6.

Junos software supports BGP for IPv6 in the following ways:

- **Native IPv6 Peering:** This method does not require any IPv4 addresses (other than a single 32-bit router ID) and supports IPv6 Network Layer Reachability Information (NLRI) only.
- **Advertising IPv6 NLRI over IPv4:** More fully leveraging multiprotocol BGP extensions, this method supports both IPv6 and IPv4 NLRI, in addition to any other needed NLRI.

In case you're wondering, IPv4 and IPv6 Network Layer Reachability Information (NLRI) are simply the destination prefix and the prefix length (network mask). They are coupled with information from the other BGP path attributes (origin, as-path, next-hop, local-pref, etc.) to produce BGP routes.

NOTE NLRI is an attribute carried by BGP update messages that was originally introduced with BGP-4 to enable (IPv4-only) support for CIDR. Today, the MP_REACH_NLRI and MP_UNREACH_NLRI attributes extend this functionality to other address families. For both IPv4 and IPv6 the NLRI is encoded in the <length, prefix> format within the BGP update message. An IPv6 example is: /32, 2001:DB8::.

Juniper Networks equipment provides many advantages when implementing BGP, and chief among them are the clear and flexible routing policy and policy subroutines found in Junos. Other advantages are the fact that IPv6 packets are forwarded in hardware and that Junos provides concise and consistent configuration language between IPv4 and IPv6.

NOTE Similar to IS-IS, the current version of BGP supports both IPv4 and IPv6, unlike the other IGP's, which require a different protocol version to support IPv6.

MORE? To learn more about BGP, NLRI, and MP-BGP, see RFC 4271: *BGP-4* (<http://www.ietf.org/rfc/rfc4271.txt>) and RFC 4760: *Multiprotocol Extensions for BGP-4* (<http://www.ietf.org/rfc/rfc4760.txt>).

Understanding Native IPv6 Peering

Native IPv6 peering establishes a BGP session over IPv6 between two IPv6-enabled BGP routers. In practical terms, this means that the configured neighbor addresses are both IPv6 addresses.

As mentioned previously, a BGP peering session established over IPv6 can only exchange IPv6 routes, which is ideal in an IPv6-only network or when you only want to exchange IPv6 routes with your BGP peer. Peering sessions over IPv6 cannot carry other NLRI, and it is rare that you would want them to, but just to cover all the bases, this chapter demonstrates native IPv6 peering on the test bed's EBGP session and shows you how to configure MP-BGP on the IBGP sessions.

BEST PRACTICE Configure all IPv6 EBGP peering sessions natively, even when you are also exchanging IPv4 routes with the same peer, to allow greater flexibility and create more redundancy than combining both NLRI on just one EBGP session.

As you should have noticed repeatedly in *Exploring IPv6*, one of the great advantages of using Junos when deploying IPv6 is how similar the configuration is to the corresponding IPv4 configuration. This fact holds true for setting up BGP peering over IPv6.

Configuring BGP with an IPv6 Peer

Using the test bed network illustrated in Figure 1.1, let's configure native IPv6 BGP between R4 and P1. If you are familiar with configuring BGP over IPv4 in Junos, these steps should probably look familiar to you.

NOTE To make this example a little more of a real-life scenario, it only configures one side of the EBGP session – you can't often configure another network's routers!

The first step is to gather the needed information (found in Figure 1.1):

- AS number: The books test bed AS number is 65000.

- Peer AS: P1 is in AS 65111.
- Neighbor IP: P1's directly connected interface address is 2001:db8:8000:4200::8/64.
- Interface IP: Choose an address in the same subnet as P1's interface, let's use 2001:db8:8000:4200::4/64.
- MD5 Key: P1 is using a key of X-L@RG3.

BEST PRACTICE Although it's not mandatory to configure a working BGP session; it is best practice to secure all BGP peering sessions with an MD5 key.

Armed with that data, let's dive right in and start configuring R4.

To Configure Native IPv6 BGP Peering:

1. First, jump to configuration mode:

```
ipv6@r4> configure
Entering configuration mode
```

2. Now set the router's AS number, under routing-options:

```
[edit]
ipv6@r4# edit routing-options

[edit routing-options]
ipv6@r4# set autonomous-system 65000
```

3. Next, configure the directly connected interface with the appropriate IPv6 address:

```
[edit routing-options]
ipv6@r4# top

[edit]
ipv6@r4# edit interfaces ge-1/0/3.84

[edit interfaces ge-1/0/3 unit 84]
ipv6@r4# set family inet6 address 2001:db8:8000:4200::4/64
```

4. Then enable BGP and create a group called PEERS by jumping to that section of the configuration:

```
[edit interfaces ge-1/0/3 unit 84]
ipv6@r4# top

[edit]
ipv6@r4# edit protocols bgp group PEERS
```

```
[edit protocols bgp group PEERS]
ipv6@r4#
```

5. Next, make this an EBGp session by setting the type to external:

```
[edit protocols bgp group PEERS]
ipv6@r4# set type external
```

6. Now configure the BGP neighbor, this includes setting the peer AS and the MD5 authentication key:

```
[edit protocols bgp group PEERS]
ipv6@r4# edit neighbor 2001:db8:8000:4200::8
```

```
[edit protocols bgp group PEERS neighbor 2001:db8:8000:4200::8]
ipv6@r4# set peer-as 65111
```

```
[edit protocols bgp group PEERS neighbor 2001:db8:8000:4200::8]
ipv6@r4# set authentication-key X-L@RG3
```

7. Finally, jump back to the top of the configuration hierarchy, verify your changes, and commit:

```
[edit protocols bgp group PEERS neighbor 2001:db8:8000:4200::8]
ipv6@r4# top
```

```
[edit]
ipv6@r4# show | compare
[edit routing-options]
+ autonomous-system 65000;
[edit protocols]
+ bgp {
+   group PEERS {
+     type external;
+     neighbor 2001:db8:8000:4200::8 {
+       authentication-key "$XDMxw2q.Pz3/9Av87-sY5Qz"; ## SECRET-DATA
+       peer-as 65111;
+     }
+   }
+ }
```

```
[edit]
ipv6@r4# commit
commit complete
```

Great! R4 now has a native IPv6 EBGp peering session configured! However, let's not start celebrating quite yet, because as it stands, this session won't do the test bed network any good. There is still an issue to resolve: R4 is not currently advertising any routes to P1, so return and inbound traffic never reaches the test bed network.

NOTE Routers running Junos advertise only routes learned via BGP (IBGP or EBGP) to EBGP peers by default.

The best way to take care of this is with an aggregate route and a routing policy, so let's configure that now.

BEST PRACTICE IPv6 routes take up more memory than IPv4 routes due to the much larger addresses (128 bit vs. 32 bit). Because of this it is even more important in IPv6 to advertise only your aggregate route outside of your network to avoid routing table "bloat."

To Configure a Basic IPv6 EBGP Export Policy:

1. Start by configuring an aggregate route for the test bed's network prefix:

```
[edit]
ipv6@r4# edit routing-options rib inet6.0 aggregate
```

```
[edit routing-options rib inet6.0 aggregate]
ipv6@r4# set route 2001:db8::/48
```

NOTE You may remember from *Day One: Exploring IPv6* that IPv6 aggregate routes must be configured in the inet6.0 rib rather than the default inet.0 rib (IPv4).

2. Next, configure the export policy that allows R4 to advertise that aggregate (and nothing else):

```
[edit routing-options rib inet6.0 aggregate]
ipv6@r4# top
```

```
[edit]
ipv6@r4# edit policy-options policy-statement peer-export
```

```
[edit policy-options policy-statement peer-export]
ipv6@r4# edit term agg
```

```
[edit policy-options policy-statement peer-export term agg]
ipv6@r4# set from protocol aggregate
```

```
[edit policy-options policy-statement peer-export term agg]
ipv6@r4# set from route-filter 2001:db8::/48 exact
```

```
[edit policy-options policy-statement peer-export term agg]
ipv6@r4# set then accept
```

```
[edit policy-options policy-statement peer-export term agg]
ipv6@r4# up
```

```
[edit policy-options policy-statement peer-export]
ipv6@r4# edit term kil
```

```
[edit policy-options policy-statement peer-export term kil]
ipv6@r4# set then reject
```

4. Now apply the policy to the PEERS BGP group:

```
[edit policy-options policy-statement peer-export term kil]
ipv6@r4# top edit protocols bgp group PEERS
```

```
[edit protocols bgp group PEERS]
ipv6@r4# set export peer-export
```

5. Then examine and commit the changes:

```
[edit protocols bgp group PEERS]
ipv6@r4# top
```

```
[edit]
ipv6@r4# show | compare
[edit routing-options]
+ rib inet6.0 {
+   aggregate {
+     route 2001:db8::/48;
+   }
+ }
[edit protocols bgp group PEERS]
+ export peer-export;
[edit policy-options]
+ policy-statement peer-export {
+   term agg {
+     from {
+       protocol aggregate;
+       route-filter 2001:db8::/48 exact;
+     }
+     then accept;
+   }
+   term kil {
+     then reject;
+   }
+ }
```

```
[edit]
ipv6@r4# commit and-quit
commit complete
Exiting configuration mode
```

NOTE The `ki1` term in our peer-export policy overrides the default BGP policy that would advertise all BGP routes. This is more important once IBGP, and/or other EBGP peers, are configured.

Now this might just be something worthwhile, but there is no way to know for sure until the operation of this brand new IPv6 EBGP peering session is verified. Read on to the next section and learn how.

MORE? If you want to learn more about configuring BGP (there is plenty more to learn) start with *Junos Software Routing Protocols Configuration Guide*, Chapters 37-39, at www.juniper.net/techpubs.

MORE? For more information on configuring routing policies in Junos, take a look at *Junos Software Policy Framework Configuration Guide, Part 2*, again found at www.juniper.net/techpubs.

Try It Yourself: Configuring Native IPv6 BGP Peering

Your turn! Use what you have learned in this section along with everything you already know about BGP to configure your test bed. Try configuring both IBGP and EBGP with native IPv6 peering. Can you set up both IPv4 and IPv6 sessions to the same peer? How about setting a prefix limit? Try experimenting with export policies to advertise other routes to your IPv6 peer. Don't forget your authentication!

Verifying Native IPv6 BGP Peering

With a native IPv6 EBGP peering session configured, it's time to focus on testing and verification. Verifying a native IPv6 BGP session in Junos is very similar to verifying an IPv4 BGP session. This section shows you how to interpret the output of the necessary commands:

- `show bgp summary`
- `show bgp neighbor`
- `show route receive-protocol`
- `show route advertising-protocol`
- `ping`

Let's get right to it.

To Verify the BGP Summary:

Begin your confirmation of the native IPv6 EBGP peering session between R4 and P1 by executing the `show bgp summary` command on R4:

```
[ipv6@r4> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed  History Damp State  Pending
inet6.0         1          1          0           0        0          0
Peer AS  InPkt  OutPkt  OutQ  Flaps Last Up/Dwn State|#Active Received /Damped...
2001:db8:8000:4200::8 65111    373    371    0     0  2:45:56 Establ
  inet6.0: 1/1/0
```

ipv6@r4>

The most important things to note here are:

- **State, Establ:** The state of our peer is established (the BGP peering session is up).
- **#Active/Received/Damped, 1/1/0:** All of the received routes are active (just one in this case).

You can tell that this is a native IPv6 session because the routing table being used is `inet6.0` and the peer is identified by an IPv6 address.

Other useful information in this output includes the time that the session last changed state (Last Up/Dwn, 2:45:56) and the peer AS number (AS, 65111).

To Examine the Details of a BGP Peer:

Now that you have confirmed that the BGP session is established, move on to take a closer look at your BGP neighbor with the `show bgp neighbor` command:

```
ipv6@r4> show bgp neighbor 2001:db8:8000:4200::8
Peer: 2001:db8:8000:4200::8+179 AS 65111 Local: 2001:db8:8000:4200::4+2304 AS 65000
Type: External State: Established Flags: <Sync>
Last State: OpenConfirm Last Event: RecvKeepAlive
Last Error: None
Export: [ peer-export ]
Options: <Preference AuthKey PeerAS Refresh>
Authentication key is configured
Holdtime: 90 Preference: 170
Number of flaps: 0
Peer ID: 10.0.1.8 Local ID: 4.4.4.4 Active Holdtime: 90
Keepalive Interval: 30 Peer index: 0
BFD: disabled, down
Local Interface: ge-1/0/3.84
```

```

NLRI advertised by peer: inet6-unicast
NLRI for this session: inet6-unicast
Peer supports Refresh capability (2)
Table inet6.0 Bit: 10000
RIB State: BGP restart is complete
Send state: in sync
Active prefixes:          1
Received prefixes:       1
Suppressed due to damping: 0
Advertised prefixes:     1
Last traffic (seconds): Received 3   Sent 22   Checked 18
Input messages:  Total 417   Updates 1   Refreshes 0   Octets 7979
Output messages: Total 415   Updates 1   Refreshes 0   Octets 7967
Output Queue[0]: 0

```

ipv6@r4>

There is a lot of good information here but let's just scan through some of the highlights. Notice first that even on this native IPv6 peering session, the router-ids are both in an IPv4 address looking format. This is because BGP requires a 32-bit router ID.

ALERT! If there are no IPv4 addresses configured on your router, you must manually assign a router ID.

BEST PRACTICE Statically assigning the router ID is considered a best practice, even in situations where it is not otherwise required

In this case R4 already has a router ID in place to support OSPF3, which you may remember also requires a 32-bit router ID:

```

ipv6@r4> show configuration routing-options router-id
router-id 4.4.4.4;

```

We can also clearly see that the only NLRI type in use is `inet6-unicast` and that the peer and local addresses at the top of the output are IPv6 addresses, confirming again that this is a native IPv6 session.

To Display BGP Routes:

Rather than viewing only the full route tables, like you would for an IGP, Junos allows you to see exactly what routes are being advertised to, and received from, each BGP peer.

1. First verify that you are receiving the expected routes from your neighbor:

```

ipv6@r4> show route receive-protocol bgp 2001:db8:8000:4200::8

inet.0: 0 destinations, 0 routes (0 active, 0 holddown, 0 hidden)

inet6.0: 29 destinations, 33 routes (29 active, 0 holddown, 0 hidden)
  Prefix          Nexthop          MED      Lc1pref  AS path
* 2001:db8:8000::/48      2001:db8:8000:4200::8          65111 I

__juniper_private1__.inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

ipv6@r4>

```

2. Now check to see that you are advertising the expected routes to your neighbor:

```

ipv6@r4> show route advertising-protocol bgp 2001:db8:8000:4200::8

inet6.0: 29 destinations, 33 routes (29 active, 0 holddown, 0 hidden)
  Prefix          Nexthop          MED      Lc1pref  AS path
* 2001:db8::/48      Self              0          0          I

ipv6@r4>

```

NOTE The `show route receive-protocol` and `show route advertising-protocol` commands must be called with a specific neighbor address, this example uses P1s address: `2001:db8:8000:4200::8`.

The output confirms quite clearly that R4 is receiving a single prefix (`2001:db8:8000::/48`) from P1 and is in turn advertising a single prefix (`2001:db8::/48`) to P1. This is exactly what you wanted to see!

To Verify Connectivity with Ping Testing:

Everything is looking great but don't stop yet. The best way to be sure that your new EBGp session actually passes traffic is by passing some! Try an old-school ping to do just that.

1. First ping an address in the peer router's advertised prefix:

```

ipv6@r4> ping 2001:db8:8000::1 count 3 rapid
PING6(56=40+8+8 bytes) 2001:db8:8000:4200::4 --> 2001:db8:8000::1
!!!
--- 2001:db8:8000::1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.278/0.342/0.455/0.080 ms

ipv6@r4>

```

2. Now source the ping from an address within your router's advertised prefix, R4s loopback:

```
ipv6@r4> ping 2001:db8:8000::1 count 3 rapid source 2001:db8::4
PING6(56=40+8+8 bytes) 2001:db8::4 --> 2001:db8:8000::1
!!!
--- 2001:db8:8000::1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.282/0.348/0.479/0.092 ms
```

```
ipv6@r4>
```

Fantastic! Sourcing the ping from an address in your advertised prefix ensures that the peer is accepting that prefix – if it wasn't, there wouldn't be a return route and the ping would fail. Based on the results seen here, this native IPv6 BGP session is in perfect working order.

MORE? Want to learn more about BGP operational mode commands? Try Chapter 3 in *Junos Software Routing Protocols and Policies Command Reference*, available at www.juniper.net/techpubs.

Try It Yourself: Verify Native IPv6 BGP Peering

Take the lessons from this section into your own test bed now, and verify your native IPv6 BGP peering sessions. Confirm that your sessions are all established and that they are advertising and receiving all of the expected IPv6 addresses. Use ping testing as a final confirmation for each prefix.

Advertising IPv6 Routes Over IPv4 Sessions

Now that you have successfully implemented native IPv6 BGP, or at least carefully followed our discussion of how to do it, it's time to try advertising IPv6 routes over an IPv4-based peering session.

There is a single primary reason to advertise IPv6 NLRI (and thus IPv6 routes) over an IPv4-based peering session rather than using native IPv6 peering: the need to advertise multiple address families (NLRI) within your IBGP topology without adding peering sessions. Generally, it is best to use native sessions unless there is a need to reduce the number of peering sessions.

An IPv4 based BGP peering session can, unlike a native IPv6 session, carry all of the NLRI supported in Junos, including:

- `inet`: IPv4
- `inet6`: IPv6
- `inet-vpn`: IPv4 Layer 3 VPNs
- `inet6-vpn`: IPv6 Layer 3 VPNs
- `l2vpn`: IPv4 MPLS-based Layer 2 VPNs and VPLS
- `inet-mdt`: IPv4 multicast distribution tree (MDT)
- `inet-mvpn`: IPv4 multicast VPNs
- `inet6-mvpn`: IPv6 multicast VPNs
- `iso-vpn`: ISO/IS-IS Layer 3 VPNs
- `route-target`: Used for VPN route filtering

Most of these address families are outside the scope of this book but those readers with MPLS and VPN experience are likely to recognize at least one or two. The main thing to take away from this list is the great flexibility gained by implementing MP-BGP over IPv4 in Junos.

ALERT! Changing the address families configured on a BGP peering session causes that session to be torn down and reestablished, resulting in a temporary disruption to the session.

Although changing the NLRI advertised over a BGP session does disrupt the peering session, your IBGP topology should be fully-redundant, making this far less disruptive than for an EBGP session. For this reason, some operators may choose to simply add the IPv6 NLRI to existing IBGP configurations. Which is exactly what is shown next!

Configuring Existing Peering to Support IPv6

Since you just configured native IPv6 BGP peering in the last section, and you have very likely configured a native IPv4 BGP peering session (or at least seen one) in the past, routers R3 through R5 in this book's test bed have been preconfigured in a full mesh IPv4-only IBGP topology. To accommodate this, the routers have all been assigned IPv4 interface addresses, which you can find in Figure 1.2. The test bed routers have also been configured with OSPFv2 to create full IPv4 reachability, in addition to the existing IPv6 connectivity.

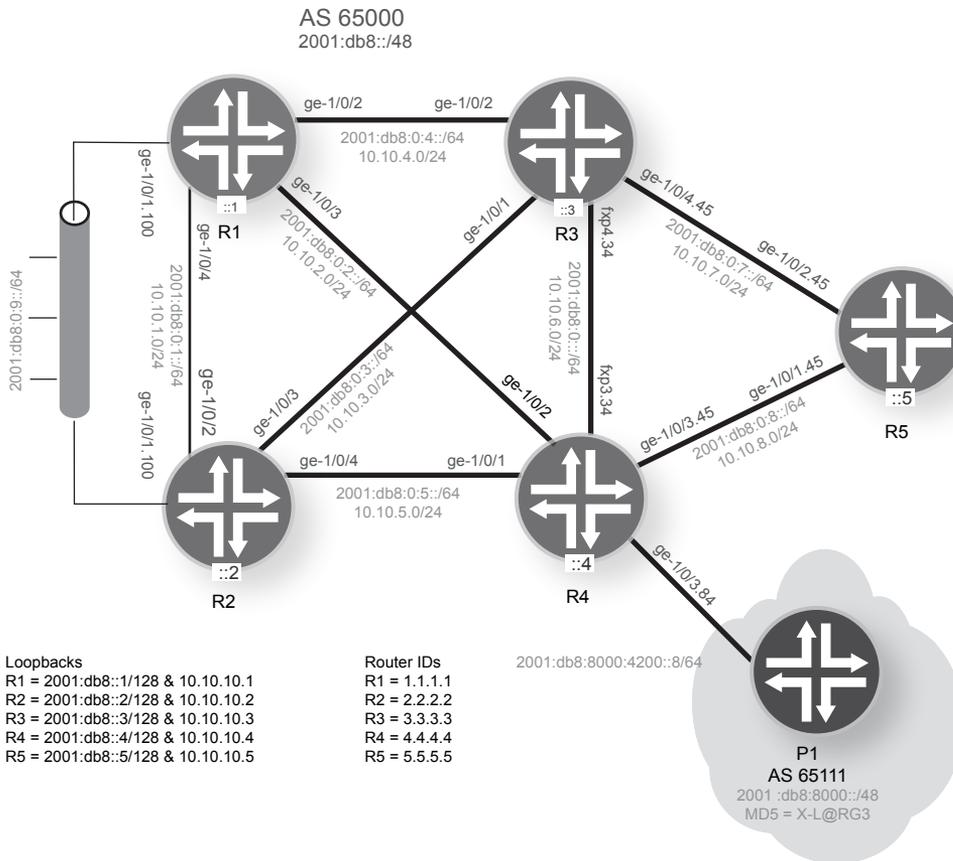


Figure 1.2 IPv6 Test Bed Topology Including IPv4 Addressing

With that in place, let's add IPv6! R4 already has an established EBGP peering session, so that's a logical place to start in the test bed network.

To Add IPv6 to an Existing IPv4 BGP Peering Session:

1. First, from configuration mode, jump to the already configured IBGP group under protocols bgp and add the inet6 address family:

```
[edit]
ipv6@r4# edit protocols bgp group IBGP
```

```
[edit protocols bgp group IBGP]
ipv6@r4# set family inet6 unicast
```

2. Next, re-add the default `inet unicast` family to this peering session:

```
[edit protocols bgp group IBGP]
ipv6@r4# set family inet unicast
```

NOTE The default `inet4 unicast` address family is disabled when any other address family (NLRI) is explicitly configured.

3. Now jump to the top, review your changes, and commit:

```
[edit protocols bgp group IBGP]
ipv6@r4# top
```

```
[edit]
ipv6@r4# show | compare
[edit protocols bgp group IBGP]
+   family inet {
+     unicast;
+   }
+   family inet6 {
+     unicast;
+   }
```

```
[edit]
ipv6@r4# commit
commit complete
```

4. Finally, use the Junos `load patch` command to configure the rest of the test bed routers. Here is what this looks like on R1:

```
ipv6@r1> configure
Entering configuration mode
```

```
[edit]
ipv6@r1# load patch terminal
[Type ^D at a new line to end input]
[edit protocols bgp group IBGP]
+   family inet {
+     unicast;
+   }
+   family inet6 {
+     unicast;
+   }
^D
load complete
```

```
[edit]
ipv6@r1# commit
commit complete
```

```
[edit]
ipv6@r1#
```

The IBGP configuration of the test bed is now set up to advertise both IPv4 and IPv6 NLRI! But, as you may have guessed, you are not done just yet. There are still two issues to resolve:

- The 2001:db8:8000:4200::/64 subnet used between R4 and P1 for eBGP peering is not being injected into the test bed's IGP. This means that you need to configure a next-hop self policy to allow the other test bed routers to resolve any routes advertised from P1.
- Additionally, IPv6 routes that are exchanged over IPv4-based iBGP peering sessions use an IPv4-mapped IPv6 address as their next hop. You must advertise R4's IPv4-mapped address into your IGP so that the other test bed routers know how to reach it.

Let's get back into R4 and ensure that our BGP next hop is correctly configured.

To Configure IPv6 Next Hop Self Over IPv4 BGP Sessions:

1. First, build a next-hop self policy for routes advertised by P1:

```
[edit]
ipv6@r4# edit policy-options policy-statement nhs

[edit policy-options policy-statement nhs]
ipv6@r4# set term P1 from protocol bgp

[edit policy-options policy-statement nhs]
ipv6@r4# set term P1 from neighbor 2001:db8:8000:4200::8

[edit policy-options policy-statement nhs]
ipv6@r4# set term P1 then next-hop self
```

2. Now add R4's IPv4-mapped IPv6 address to lo0.0 (since lo0.0 is configured in both OSPF and OSPF3, any address added here will be advertised to the other test-bed routers):

```
[edit policy-options policy-statement nhs]
ipv6@r4# top

[edit]
ipv6@r4# edit interfaces lo0 unit 0

[edit interfaces lo0 unit 0]
ipv6@r4# set family inet6 address ::ffff:10.10.10.4
```

NOTE You should know to use ::ffff:10.10.10.4 because 10.10.10.4 is R4's IPv4 lo0 address and also the local-address configured in group IBGP.

3. Then, ensure deterministic behavior for sessions to lo0 by setting one of your addresses as primary and preferred:

```
[edit interfaces lo0 unit 0]
ipv6@r4# set family inet6 address 2001:db8::4/128 primary preferred
```

4. Next, apply the next-hop self policy to your IBGP peering sessions as an export policy:

```
[edit interfaces lo0 unit 0]
ipv6@r4# top

[edit]
ipv6@r4# edit protocols bgp group IBGP

[edit protocols bgp group IBGP]
ipv6@r4# set export nhs
```

5. Now verify your changes and commit:

```
[edit protocols bgp group IBGP]
ipv6@r4# top

[edit]
ipv6@r4# show | compare
[edit interfaces lo0 unit 0 family inet6 address 2001:db8::2/128]
+   primary;
+   preferred;
[edit interfaces lo0 unit 0 family inet6]
+   address 2001:db8::4/128 { ... }
+   address ::ffff:10.10.10.4/128;
[edit protocols bgp group IBGP]
+   export nhs;
[edit policy-options]
+   policy-statement nhs {
+     term P1 {
+       from {
+         protocol bgp;
+         neighbor 2001:db8:8000:4200::8;
+       }
+       then {
+         next-hop self;
+       }
+     }
+   }

[edit]
ipv6@r4# commit
commit complete

[edit]
ipv6@r4#
```

Okay! Now it's time to start verifying IPv6 support in the test bed's IBGP topology.

BEST PRACTICE In this book's test bed network, R4 is the only router advertising IPv6 prefixes, so it is not absolutely necessary to assign and advertise IPv4-mapped addresses on the other test bed routers. It is, however, best practice that you preconfigure this up front, since it's likely that at least some other routers in your network will advertise IPv6 routes into your IBGP at some point.

MORE? If you want to learn more about advertising IPv6 prefixes over IPv4-based BGP sessions, see "Enabling Multiprotocol BGP" and "Configuring IPv6 BGP Routes over IPv4 Transport" in Chapter 38 of *Junos Software Routing Protocols Configuration Guide*, which can be found at www.juniper.net/techpubs.

Try It Yourself: Configuring IPv4 IBGP to Support IPv6

It's that time again, time for you to take what you just learned in this section and put it to use! Set up IBGP in your own test network to support IPv4 and IPv6 over the same peering sessions. Try injecting more routes and adding more address families as well. Can you filter specific routes? How about combining IPv4 and IPv6 filters in the same policy? Since adding the inet6 family causes your IBGP sessions to reestablish, this is a great time to add authentication if it wasn't previously configured! Remember to add the IPv4-mapped addresses!

Remember to share what you did and what you found interesting by creating a post on this book's pages on J-Net at www.juniper.net/dayone.

Verifying IBGP Peering and IPv6 Support

Bring on the verification! Because Junos has implemented IPv6 so elegantly, the steps for verifying IPv6 support over IPv4-based sessions are very similar to the steps you took to verify native IPv6 peering. Let's start verification in the book's test bed on R1.

To Verify IPv6 Support Over IPv4-based BGP Peering:

1. First, confirm that all IBGP sessions are established:

```

ipv6@r1> show bgp summary
Groups: 1 Peers: 4 Down peers: 0
Table      Tot Paths  Act Paths  Suppressed  History  Damp State  Pending
inet.0     0          0          0           0        0         0
inet6.0    1          1          0           0        0         0

```

Peer	AS	InPkt	OutPkt	OutQ	Flaps	Last Up/Dwn	State #Active/
Received/Damped...							
10.10.10.2	65000	70	72	0	0	31:57	Establ
inet.0: 0/0/0							
inet6.0: 0/0/0							
10.10.10.3	65000	69	71	0	0	31:53	Establ
inet.0: 0/0/0							
inet6.0: 0/0/0							
10.10.10.4	65000	72	71	0	0	31:57	Establ
inet.0: 0/0/0							
inet6.0: 1/1/0							
10.10.10.5	65000	71	73	0	0	32:00	Establ
inet.0: 0/0/0							
inet6.0: 0/0/0							

2. Next look for any hidden routes:

```
ipv6@r1> show route hidden
```

```
inet.0: 19 destinations, 19 routes (19 active, 0 holddown, 0 hidden)
```

```
inet6.0: 27 destinations, 30 routes (27 active, 0 holddown, 0 hidden)
```

```
__juniper_private1__inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
```

3. Then look over the BGP neighbor details; here we examine the session with R3:

```
ipv6@r1> show bgp neighbor 10.10.10.3
```

```
Peer: 10.10.10.3+179 AS 65000 Local: 10.10.10.1+1138 AS 65000
Type: Internal State: Established Flags: <Sync>
Last State: OpenConfirm Last Event: RecvKeepAlive
Last Error: None
Options: <Preference LocalAddress AuthKey AddressFamily Refresh>
Authentication key is configured
Address families configured: inet-unicast inet6-unicast
Local Address: 10.10.10.1 Holdtime: 90 Preference: 170
Number of flaps: 0
Peer ID: 3.3.3.3 Local ID: 1.1.1.1 Active Holdtime: 90
Keepalive Interval: 30 Peer index: 3
BFD: disabled, down
NLRI advertised by peer: inet-unicast inet6-unicast
NLRI for this session: inet-unicast inet6-unicast
Peer supports Refresh capability (2)
Table inet.0 Bit: 10000
RIB State: BGP restart is complete
Send state: in sync
Active prefixes: 0
Received prefixes: 0
Suppressed due to damping: 0
Advertised prefixes: 0
```

```

Table inet6.0 Bit: 20000
  RIB State: BGP restart is complete
  Send state: in sync
  Active prefixes:          0
  Received prefixes:       0
  Suppressed due to damping: 0
  Advertised prefixes:     0
Last traffic (seconds): Received 10 Sent 5 Checked 55
Input messages: Total 80 Updates 0 Refreshes 0 Octets 1520
Output messages: Total 82 Updates 0 Refreshes 0 Octets 1592
Output Queue[0]: 0
Output Queue[1]: 0

```

The most important things to note in this output are:

- **NLRI / Address families:** Both inet-unicast and inet6-unicast are listed under the output parameters: Address families configured, NLRI advertised by peer, and, NLRI for this session.
- **Route Tables:** Both the inet.0 and inet.6 RIBs are listed.

4. Now verify that R1 is receiving BGP routes:

```

ipv6@r1> show route protocol bgp

inet.0: 19 destinations, 19 routes (19 active, 0 holddown, 0 hidden)
inet6.0: 27 destinations, 30 routes (27 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2001:db8:8000::/48 *[BGP/170] 00:52:42, localpref 100, from 10.10.10.4
  AS path: 65111 I
    > to fe80::2a0:c9ff:feca:9cc2 via ge-1/0/3.0

__juniper_private1__.inet6.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)

```

This book's test bed network only has one EBGP peer and thus only one BGP route.

5. Next, take a closer look at P1's route:

```

ipv6@r1> show route 2001:db8:8000::/48 detail

inet6.0: 27 destinations, 30 routes (27 active, 0 holddown, 0 hidden)
2001:db8:8000::/48 (1 entry, 1 announced)
  *BGP Preference: 170/-101
  Next-hop reference count: 3
  Source: 10.10.10.4
  Next hop: fe80::2a0:c9ff:feca:9cc2 via ge-1/0/3.0, selected

```

```

Protocol next hop: ::ffff:10.10.10.4
Indirect next hop: 87839a0 131070
State: <Active Int Ext>
Local AS: 65000 Peer AS: 65000
Age: 56:12 Metric2: 1
Task: BGP_65000.10.10.10.4+2563
Announcement bits (2): 0-KRT 4-Resolve tree 2
AS path: 65111 I Aggregator: 65111 10.0.1.8
Localpref: 100
Router ID: 4.4.4.4

```

You can clearly see that this route is being advertised by R4 due to the Source and Router ID fields. Also, take a look at the Protocol next hop field and you will see that the next hop self policy and IPv4-mapped address that were added to R4 are being put to good use.

6. Now use a sourced ping to verify connectivity into AS 65111:

```

ipv6@r1> ping 2001:db8:8000::1 source 2001:db8::1 rapid count 3
PING6(56=40+8+8 bytes) 2001:db8::1 --> 2001:db8:8000::1
!!!
--- 2001:db8:8000::1 ping6 statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.369/0.450/0.585/0.096 ms

```

7. Finally, use traceroute to verify that the best path to AS 65111 is being used:

```

ipv6@r1> traceroute 2001:db8:8000::1
traceroute6 to 2001:db8:8000::1 (2001:db8:8000::1) from 2001:db8:0:2::1, 30 hops max,
12 byte packets
 1 2001:db8:0:2::4 (2001:db8:0:2::4) 0.297 ms 0.268 ms 0.147 ms
 2 2001:db8:8000::1 (2001:db8:8000::1) 0.501 ms 0.478 ms 0.427 ms

```

```

ipv6@r1>

```

Take a deep breath; you have now fully configured a functional IPv6 BGP network!

BEST PRACTICE Conduct all of the verification steps listed here across all of your network's IBGP routers to ensure a fully functioning deployment.

MORE? As mentioned in the native IPv6 verification section, more on Junos operational mode commands for verifying BGP can be found in Chapter 3 of *Junos Software Routing Protocols and Policies Command Reference*, available at www.juniper.net/techpubs.

Try It Yourself: Verify IPv6 Support with IPv4-based BGP Peering

Now get into your own test bed network and verify IPv6 support on your IPv4-based BGP peering sessions. Confirm that your sessions are all established and that they are advertising and receiving all of the expected IPv6 prefixes. Look for hidden routes and then use ping and traceroute testing to confirm reachability for each prefix.

Chapter 2

Getting Ready for Production IPv6

<i>Exploring DHCPv6.....</i>	<i>30</i>
<i>Introducing VRRP for IPv6</i>	<i>32</i>
<i>Understanding CoS in IPv6</i>	<i>39</i>
<i>Introducing Multicast Listener Discovery.....</i>	<i>49</i>

At this point, the book's test bed network is fully configured from a routing protocol perspective, but as you probably know, there is more to operating a successful network than just exchanging routes. This chapter explores some of the necessary protocols and tools needed to run your new IPv6 network.

This chapter discusses DHCPv6, VRRP for IPv6, and CoS in IPv6. By chapter's end you will learn how all of these often essential network building blocks are different from their IPv4 counterparts, as well as how to configure and verify both VRRP and CoS for IPv6.

Exploring DHCPv6

Day One: Exploring IPv6 introduced and explained SLAAC (Stateless Address Auto Configuration), the lightweight address assignment mechanism introduced with IPv6's Neighbor Discovery protocol. In addition to SLAAC, IPv6 maintains DHCP through the new DHCPv6 protocol, which we'll now explore.

ALERT! You should have a good understanding of the Junos implementation of DHCP in IPv4 before tackling this section.

MORE? For more information on DHCP for IPv4 in Junos, see *Junos Software Subscriber Access Configuration Guide* and *Junos Software Broadband Subscriber Management Solutions Guide*, available at <http://www.juniper.net/techpubs>.

Diving Into DHCPv6

Why would you want to use DHCPv6 if autoconfiguration is built into IPv6? Well, the first thing to know about dynamic host configuration in IPv6 is that DHCPv6 and SLAAC are compatible and can be used together. There are two general reasons to implement DHCPv6 either in place of, or in addition to, SLAAC:

- To provide more information: While SLAAC provides a limited amount of information about the network, DHCPv6 provides clients with DNS server addresses and search options, SNTP servers, NIS configuration, and more, all in addition to IPv6 addresses and prefixes.

- To provide more control: Some operators may prefer to have control over which addresses are allocated or which clients are allowed to receive network information. DHCPv6 provides this control. DHCPv6 also provides visibility into which clients hold which addresses, and when. This is very useful for auditing.

Once you have decided to use DHCPv6, you should understand how it differs from DHCP in IPv4. A few of the key highlights include:

- New protocol: While DHCP for IPv4 is based on BOOTP, an older protocol, DHCPv6 is a brand new purpose-built protocol, that leaves behind many of the inefficiencies found in DHCP for IPv4.
- Multicast: Like many other IPv6 protocols, DHCPv6 uses multicast, rather than broadcast.
- Link-local addresses: DHCPv6 clients use link local addresses to send their requests.
- Single exchange: A single DHCPv6 request can include every interface on a client, allowing the server to provide addresses for all interfaces in a single exchange.
- Statefull or stateless: DHCPv6 can operate in a statefull or stateless mode. Statefull is used when you need to allocate IPv6 addresses using DHCPv6, much like DHCP for IPv4. Stateless is used when another method (such as SLAAC) is used to provide IPv6 addresses and you only need DHCPv6 to provide other configuration information.

MORE? For more information on DHCPv6, look at *RFC 3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* at <http://tools.ietf.org/rfc/rfc3315.txt>.

Configuring DHCPv6

Due to time and equipment constraints while writing this book, it was impossible to provide a complete reference configuration for DHCPv6 in Junos. DHCPv6 implementations are still in a state of flux and due to that there are still enough moving parts to make it impossible for this book to provide a simple reference configuration that will definitely work in your situation. It is the nature of dynamic migration that some elements are being worked on while they are being documented.

What can be done is to point you in the right direction. At the time of this writing, there are four primary pieces to a DHCPv6 configuration in Junos:

1. The DHCPv6 local server group: Configured under `system services dhcp-local-server dhcpv6`.
2. The subscriber management dynamic profile: Configured under `dynamic-profiles`.
3. RADIUS authentication: Configured under `access radius-server` and `access profile`.
4. The inet6 address assignment pool: Configured under `access address-assignment`.

Essentially, that makes this section a large *Try It Yourself* section! Go forth and test DHCPv6! As you find solutions that work (and that do not), please submit your experiences on the book's J-Net pages at www.juniper.net/dayone. The author intends to do so. And if DHCPv6 configuration is critical to you, you might check the J-Net boards for real-world implementation discussion.

TIP Depending on when you are reading this book, there may be an updated version of this book on www.juniper.net/dayone. If the DHCPv6 configuration is critical to you, it might be worth your time to check for an update.

MORE? For more on configuring DHCP in Junos, see *Junos Software Subscriber Access Configuration Guide* and *Junos Software Broadband Subscriber Management Solutions Guide*, both available at www.juniper.net/techpubs.

Introducing VRRP for IPv6

Virtual Router Redundancy Protocol (VRRP) allows hosts on a LAN to make use of redundant routing platforms without any additional configuration on the hosts beyond a single default route.

The VRRP routers share the IP address, which corresponds to the LAN hosts' default route. At any time, one of the VRRP routers is active (the Master) and the others are backups. The Master router accepts and forwards packets sent to the virtual routers' IP address (the address

configured in the hosts default route). The Master also sends advertisements to the backup routers at regular intervals (the default is one second).

If a backup router does not receive any advertisements from the Master for a set period of time, the backup router with the next highest priority becomes the Master and takes over forwarding packets. This dynamic failover is done with very little VRRP traffic and without any interaction from the hosts.

VRRP for IPv6 is the third version of the protocol and it varies from version two in three key ways:

- IPv6 Support: Not found in version two.
- No Authentication: IPv4-only VRRP protocol exchanges can be authenticated, but this option is not available when using VRRP for IPv6.
- Link-local address: VRRP for IPv6 requires that you explicitly define a virtual link-local address for each group. The virtual link-local address must be on the same subnet as the physical interface address.

If you are familiar with IPv6, you may be asking yourself, “Can’t I use IPv6 Neighbor Discovery (ND) to provide router redundancy for a LAN?”

The answer is: Yes, but VRRP for IPv6 provides a much faster failover to the backup router than IPv6 ND. This faster switchover makes VRRP for IPv6 the preferred choice for many operators.

MORE? For all the details on VRRP for IPv6, check out *draft-ietf-rrp-ipv6-spec-08.txt: Virtual Router Redundancy Protocol for IPv6* at <http://tools.ietf.org/id/draft-ietf-rrp-ipv6-spec-08.txt>.

Configuring VRRP for IPv6

In this book’s test bed network, R1 and R2 are configured as VRRP routers to provide redundancy to their connected LAN. The virtual router group uses IP address 2001:db8:0:9::1 and R1 will be the preferred Master. This is illustrated in Figure 2.1:

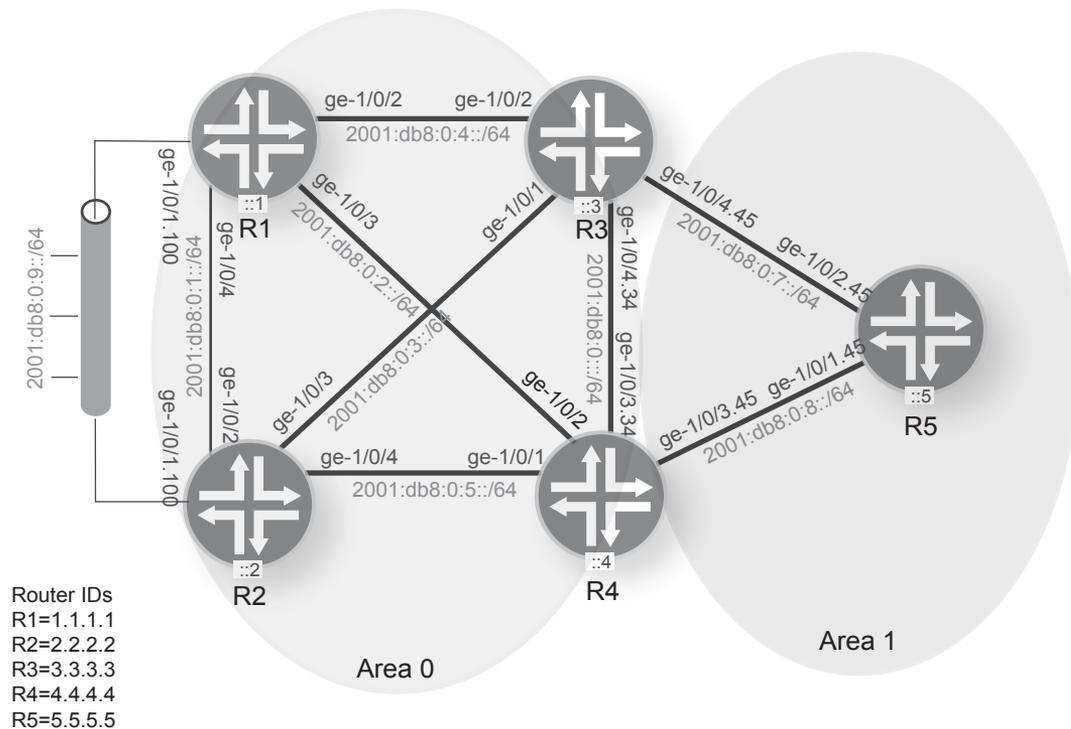


Figure 2.1 VRRP for IPv6 Test Bed Topology

All clients on the LAN are already configured with a default gateway IP address of 2001:db8:0:9::1. So let's start by configuring R1.

To Configure VRRP for IPv6 on R1:

1. First enable Router Advertisements on the LAN facing interface:

```
[edit]
ipv6@r1# edit protocols router-advertisement interface ge-1/0/1.100
```

```
[edit protocols router-advertisement interface ge-1/0/1.100]
ipv6@r2# set prefix 2001:db8:0:9::/64
```

2. Then explicitly specify the link-local address on that interface:

```
[edit protocols router-advertisement]
ipv6@r1# top edit interfaces ge-1/0/1 unit 100 family inet6
```

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r1# set address fe80:db8:0:9::3/64
```

3. Now create the VRRP for IPv6 virtual router group under address 2001:db8:0:9::3/64:

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r1# edit address 2001:db8:0:9::3/64 vrrp-inet6-group 42
```

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1#
```

4. Next add the virtual router group's IP address:

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1# set virtual-inet6-address 2001:db8:0:9::1
```

5. Then configure the virtual router group's link-local address:

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1# set virtual-link-local-address fe80:db8:0:9::1
```

6. Now add a priority and ensure that R1 is always the Master when available by setting preempt:

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1# set priority 250
```

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1# set preempt
```

7. Finally, configure accept-data so that the LAN hosts can ping the virtual router IP (their default gateway):

```
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64 vrrp-inet6-
group 42]
ipv6@r1# set accept-data
```

We're halfway there! R1 now has VRRP for IPv6 configured.

BEST PRACTICE

When using the accept-data command you should configure firewall filters to accept only ICMP packets.

In order to provide redundancy to the LAN, the test bed needs at least one more VRRP router. R2 is up next and will go much more quickly if a couple of handy tools are leveraged in Junos.

To Configure VRRP for IPv6 on R2:

1. Before you commit your changes on R1, issue a `show | compare` command:

```
[edit]
ipv6@r1# show | compare
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::3/64]
+   vrrp-inet6-group 42 {
+     virtual-inet6-address 2001:db8:0:9::1;
+     virtual-link-local-address fe80:db8:0:9::1;
+     priority 250;
+     preempt;
+     accept-data;
+   }
[edit interfaces ge-1/0/1 unit 100 family inet6]
  address 2001:db8:0:9::3/64 { ... }
+   address fe80:db8:0:9::3/64;
[edit]
+ protocols {
+   router-advertisement {
+     interface ge-1/0/1.100 {
+       prefix 2001:db8:0:9::/64;
+     }
+   }
+ }
```

2. Now copy the output into a text editor, change the physical interface addresses from `::3` to `::2` and lower the priority (from 250 to 200).

3. Then, enter the edited text into R2 using `load patch terminal`:

```
[edit]
ipv6@r2# load patch terminal
[Type ^D at a new line to end input]
[edit interfaces ge-1/0/1 unit 100 family inet6 address 2001:db8:0:9::2/64]
+   vrrp-inet6-group 42 {
+     virtual-inet6-address 2001:db8:0:9::1;
+     virtual-link-local-address fe80:db8:0:9::1;
+     priority 200;
+     preempt;
+     accept-data;
+   }
[edit interfaces ge-1/0/1 unit 100 family inet6]
  address 2001:db8:0:9::2/64 { ... }
+   address fe80:db8:0:9::2/64;
[edit]
+ protocols {
+   router-advertisement {
+     interface ge-1/0/1.100 {
```

```

+           prefix 2001:db8:0:9::/64;
+           }
+       }
+ }
^D
load complete

```

Once the changes have been committed on both routers it's time to verify that VRRP for IPv6 is operating properly, which is our next task.

MORE? To learn more about configuring VRRP for IPv6 in Junos, take a look at *Junos Software High Availability Configuration Guide*, available at www.juniper.net/techpubs.

Try It Yourself: Configuring VRRP for IPv6

Ah, the fun part! Jump into your own test bed network and configure VRRP for IPv6 using what you just learned. Remember to enable Router Advertisements and to hard-set your link-local addresses. Play with priority, preemption, and accept-data. Try configuring more than two VRRP routers. Can you use interface and route tracking to dynamically change VRRP priorities? Tell us the answer on this book's postings on J-Net: www.juniper.net/dayone.

Verifying VRRP for IPv6

Like any networking protocol, VRRP for IPv6 is only useful when it's operating as expected. In order to know this before your users complain, you must verify!

To Verify VRRP for IPv6:

1. First, ensure that VRRP for IPv6 is running on R1 and that R1 is the Master:

```

[ipv6@r1> show vrrp
Interface      State
ge-1/0/1.100  up

```

Group	VR state	VR Mode	Timer	Type	Address
42	master	Active	A 0.198	1c1	2001:db8:0:9::3
				vip	fe80:db8:0:9::1
				vip	2001:db8:0:9::1

2. Next, check R2 to see that it is a backup for VRRP group 42:

```

ipv6@r2> show vrrp
Interface      State
ge-1/0/1.100  up

```

Group	VR state	VR Mode	Timer	Type	Address
42	backup	Active	D 2.285	1c1	2001:db8:0:9::2
				vip	fe80:db8:0:9::1
				vip	2001:db8:0:9::1
				mas	fe80:db8:0:9::3

3. Now confirm that the VRRP options are properly set on both routers. Here is the output from R1:

```
ipv6@r1> show vrrp detail
Physical interface: ge-0/0/0, Unit: 666, Vlan-id: 666, Address: 2001:db8:0:9::3/64
Index: 70, SNMP ifIndex: 133, VRRP-Traps: disabled
Interface state: up, Group: 42, State: master, VRRP Mode: Active
Priority: 250, Advertisement interval: 1, Authentication type: none
Delay threshold: 100, Computed send rate: 0
Preempt: yes, Accept-data mode: yes, VIP count: 2, VIP: fe80:db8:0:9::1,
2001:db8:0:9::1
Advertisement Timer: 0.004s, Master router: fe80:db8:0:9::3
Virtual router uptime: 00:22:36, Master router uptime: 00:17:47
Virtual Mac: 00:00:5e:00:02:2a
Tracking: disabled
```

Here you can clearly see that preempt and accept-data are both set to yes, that tracking is disabled, and that R1's priority is 250.

4. Next, deactivate R1's interface to test failover:

```
[edit]
ipv6@r1# deactivate interfaces ge-1/0/1 unit 100
```

```
[edit]
ipv6@r1# commit
commit complete
```

5. Then ensure that R2 has taken over as Master:

```
ipv6@r2> show vrrp
Interface      State      Group  VR state VR Mode  Timer  Type  Address
ge-1/0/1.100  up         42    master  Active  A 0.271 lc1 2001:db8:0:9::2
                vip      fe80:db8:0:9::1
                vip      2001:db8:0:9::1
```

6. Now re-activate ge-1/0/1.100 on R1 and verify that it preempts R2:

```
[edit]
ipv6@r1# rollback 1
load complete
```

```
[edit]
ipv6@r1# commit and-quit
commit complete
Exiting configuration mode
```

```
ipv6@r1> show vrrp
Interface      State      Group  VR state VR Mode  Timer  Type  Address
ge-1/0/1.100  up         42    master  Active  A 0.282 lc1 2001:db8:0:9::3
                vip      fe80:db8:0:9::1
                vip      2001:db8:0:9::1
```

7. Finally, ping the Virtual IP from one of the LAN hosts, to ensure that accept-data is working:

```
client@HOST1> ping6 2001:db8:0:9::1
PING 2001:db8:0:9::1(2001:db8:0:9::1) 56 data bytes
64 bytes from 2001:db8:0:9::1: icmp_seq=0 ttl=64 time=7.01 ms
64 bytes from 2001:db8:0:9::1: icmp_seq=1 ttl=64 time=2.84 ms
64 bytes from 2001:db8:0:9::1: icmp_seq=2 ttl=64 time=2.27 ms
64 bytes from 2001:db8:0:9::1: icmp_seq=3 ttl=64 time=2.25 ms
^C
--- 2001:db8:0:9::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 2.253/3.598/7.018/1.988 ms, pipe 2
```

Fantastic! Our test bed's LAN has router redundancy!

MORE? For more on verifying VRRP for IPv6, see the Junos Software Interfaces Command Reference, available at www.juniper.net/techpubs.

Try It Yourself: Verify VRRP for IPv6

Now take these lessons and verify VRRP for IPv6 in your own test bed network. What happens when the configured options don't match in all the VRRP routers? Use tracking and verify the priority changes when routes or interfaces being tracked go up and down. How long does failover take? Can you improve that time?

Understanding CoS in IPv6

Class of Service (CoS) is a set of tools and procedures for applying specific levels of service to different classes of traffic. Network operators can leverage CoS features in their Juniper routers to define service levels which provide different delay, jitter, and packet loss characteristics to each traffic class.

In order for a CoS deployment to be effective, it must work consistently through the entire network, end-to-end. In a multi-vendor environment, like those found in many production networks, this requires a vendor neutral, standards-based implementation of CoS on all network equipment. Juniper Networks routers interoperate with other vendors' devices because they are based on IETF Differentiated Services (DiffServ) standards.

DiffServ specifies a six-bit field in the IP header, which is used to signal which service class that packet belongs in. The values carried in this field are DiffServ Code Points (DSCPs) and are set by the host applica-

tion or by a router at the edge of the network. All other routers along the packet's path use the DSCP value to determine how the packet should be treated when forwarding traffic.

The primary difference between DiffServ for IPv4 and IPv6 is the location of the DSCP bits in the packet header. Junos facilitates this change with the introduction of a new configuration and command statement: `dscp-ipv6`. In the next sections, you will learn how to take advantage of this new statement in order to configure CoS for IPv6.

ALERT! You should have a solid understanding of CoS in IPv4 before tackling the following sections.

MORE? To learn more about CoS in IPv6, take a look at *RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers* and *RFC 4594: Configuration Guidelines for DiffServ Service Classes* both available at <http://tools.ietf.org/>.

Configuring CoS in IPv6

CoS is a vast and often complicated subject that could easily fill several Day One books on its own, but this book will try to focus specifically on the differences between configuring and verifying CoS in IPv6 and in IPv4. Once you've finished reading this section, you should have the tools necessary to apply your knowledge of CoS in IPv4 to your IPv6 network.

Junos allows enormous flexibility through a number of CoS mechanisms that can be configured individually, or in combination, to provide a multitude of individual service offerings. These mechanisms are:

- **Classifiers:** The two types of classifiers that allow you to associate incoming packets with a forwarding class and a Packet Loss Priority (PLP) are Behavior Aggregate (BA), which uses DSCP and will be covered here, and Multi-Field (MF), which uses firewall filters for greater flexibility.
- **Forwarding classes:** These allow you to define the marking and scheduling of packets as they transit the router. Forwarding classes combine with the PLP to determine the router's Per-Hop Behavior (PHB) for CoS.

- **Loss priorities:** Just as the name implies, this allows you to specify the PLP for a specific class of traffic.
- **Forwarding policy options:** These allow you to define CoS-based forwarding. This means associating traffic classes with next-hops.
- **Schedulers:** Allow you to control packet transmission. Options include priority, bandwidth, delay buffer size, rate control status, and Random Early Detection (RED) drop profiles, per traffic class.
- **Policers:** Applied to either input or output interfaces, these allow you to limit the traffic in a specified class to a configured bandwidth and burst size; options for traffic exceeding the limit include discarding and re-classifying.
- **Rewrite markers:** These allow you to rewrite the DSCP value on outgoing packets.

The `dscp-ipv6` configuration statement is only needed when configuring rewrite rules and BA classifiers, so let's build a CoS policy on R2 to see what that looks like.

To Configure a CoS Policy for IPv6:

1. Begin by creating a classifier, call it `my-dscp-map`:

```
[edit]
ipv6@r2# edit class-of-service classifiers dscp-ipv6 my-dscp-map
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map]
ipv6@r2#
```

2. Then define the forwarding classes, the book's test bed will use three:

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map]
ipv6@r2# edit forwarding-class best-effort
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class best-effort]
ipv6@r2# set loss-priority low code-points 000000
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class best-effort]
ipv6@r2# up
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map]
ipv6@r2# edit forwarding-class assured-forwarding
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class assured-forwarding]
```

```
ipv6@r2# set loss-priority low code-points [ 001010 010100 ]
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class assured-forwarding]
```

```
ipv6@r2# up
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map]
```

```
ipv6@r2# edit forwarding-class network-control
```

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class network-control]
```

```
ipv6@r2# set loss-priority low code-points 110000
```

3. Next, configure the dscp-ipv6 rewrite rules for each forwarding class:

```
[edit class-of-service classifiers dscp-ipv6 my-dscp-map forwarding-class network-control]
```

```
ipv6@r2# up 3
```

```
[edit class-of-service]
```

```
ipv6@r2# edit rewrite-rules dscp-ipv6 my-dscp-rewrite
```

```
[edit class-of-service rewrite-rules dscp-ipv6 my-dscp-rewrite]
```

```
ipv6@r2# set forwarding-class best-effort loss-priority low code-point 000000
```

```
[edit class-of-service rewrite-rules dscp-ipv6 my-dscp-rewrite]
```

```
ipv6@r2# set forwarding-class assured-forwarding loss-priority low code-point 001010
```

```
[edit class-of-service rewrite-rules dscp-ipv6 my-dscp-rewrite]
```

```
ipv6@r2# set forwarding-class network-control loss-priority low code-point 110000
```

4. Now apply the new classifier and rewrite rule to an interface:

```
[edit class-of-service rewrite-rules dscp-ipv6 my-dscp-rewrite]
```

```
ipv6@r2# up 2
```

```
[edit class-of-service]
```

```
ipv6@r2# edit interfaces ge-1/0/1 unit 100
```

```
[edit class-of-service interfaces ge-1/0/1 unit 100]
```

```
ipv6@r2# set classifiers dscp-ipv6 my-dscp-map
```

```
[edit class-of-service interfaces ge-1/0/1 unit 100]
```

```
ipv6@r2# set rewrite-rules dscp-ipv6 my-dscp-rewrite
```

5. Finish by verifying your configuration changes and committing:

```
[edit class-of-service interfaces ge-1/0/1 unit 100]
```

```
ipv6@r2# top

[edit]
ipv6@r2# show | compare
[edit]
+ class-of-service {
+   classifiers {
+     dscp-ipv6 my-dscp-map {
+       forwarding-class assured-forwarding {
+         loss-priority low code-points [ 001010 010100 ];
+       }
+       forwarding-class network-control {
+         loss-priority low code-points 110000;
+       }
+       forwarding-class best-effort {
+         loss-priority low code-points 000000;
+       }
+     }
+   }
+   interfaces {
+     ge-0/0/0 {
+       unit 666 {
+         classifiers {
+           dscp-ipv6 my-dscp-map;
+         }
+         rewrite-rules {
+           dscp-ipv6 my-dscp-rewrite;
+         }
+       }
+     }
+   }
+   rewrite-rules {
+     dscp-ipv6 my-dscp-rewrite {
+       forwarding-class best-effort {
+         loss-priority low code-point 000000;
+       }
+       forwarding-class assured-forwarding {
+         loss-priority low code-point 001010;
+       }
+       forwarding-class network-control {
+         loss-priority low code-point 110000;
+       }
+     }
+   }
+ }

[edit]
ipv6@r2# commit
commit complete
```

Not too shabby. R2 now has a complete CoS policy applied to ge-1/0/1.100, which will classify incoming IPv6 traffic and ensure that outgoing IPv6 traffic is properly marked.

NOTE You can apply both `dscp` and `dscp-ipv6` classifiers and re-write rules on the same interface.

The other piece of your CoS configuration that will look different for IPv6 is the Class-Based Forwarding (CBF) policy. The next tutorial illustrates this.

To Configure CoS-based Forwarding for IPv6:

1. First, create the CBF next-hop map:

```
[edit]
ipv6@r2# edit class-of-service forwarding-policy next-hop-map my-cbf-map
```

```
[edit class-of-service forwarding-policy next-hop-map my-cbf-map]
ipv6@r2#
```

2. Now define the next-hops so that BE traffic is sent to R3 and AF traffic is forwarded to R4:

```
[edit class-of-service forwarding-policy next-hop-map my-cbf-map]
ipv6@r2# set forwarding-class best-effort next-hop 2001:db8:0:3::3
```

```
[edit class-of-service forwarding-policy next-hop-map my-cbf-map]
ipv6@r2# set forwarding-class assured-forwarding next-hop 2001:db8:0:5::4
```

3. Next configure a forwarding policy that applies the next-hop map to all IPv6 traffic:

```
[edit class-of-service forwarding-policy next-hop-map my-cbf-map]
ipv6@r2# top edit policy-options policy-statement CBF
```

```
[edit policy-options policy-statement CBF]
ipv6@r2# set term v-six from family inet6
```

```
[edit policy-options policy-statement CBF]
ipv6@r2# set term v-six then cos-next-hop-map my-cbf-map
```

4. Then apply the forwarding policy:

```
[edit policy-options policy-statement CBF]
ipv6@r2# top edit routing-options forwarding-table
```

```
[edit routing-options forwarding-table]
ipv6@r2# set export CBF
```

5. Finally, look over your changes and commit:

```
[edit routing-options forwarding-table]
ipv6@r2# top

[edit]
ipv6@r2# show | compare
[edit]
+ routing-options {
+   forwarding-table {
+     export CBF;
+   }
+ }
+ policy-options {
+   policy-statement CBF {
+     term v-six {
+       from family inet6;
+       then cos-next-hop-map my-cbf-map;
+     }
+   }
+ }
[edit class-of-service]
+ forwarding-policy {
+   next-hop-map my-cbf-map {
+     forwarding-class best-effort {
+       next-hop 2001:db8:0:3::3;
+     }
+     forwarding-class assured-forwarding {
+       next-hop 2001:db8:0:5::4;
+     }
+   }
+ }

[edit]
ipv6@r2# commit
commit complete
```

If you are an experienced CBF implementer you have likely noticed that the only differences between this configuration and an IPv4 CBF Junos config is the use of an IPv6 next-hop and the use of `family inet6` in the forwarding policy.

In fact, applying the policy to only IPv6 through the `from family inet6` statement is not required. You can omit that and configure both IPv6 and IPv4 next-hops in your next-hop map, if that makes sense in your implementation. In that case, the configuration would look something like this:

```
routing-options {
  forwarding-table {
    export CBF;
```

```

    }
  }
  policy-options {
    policy-statement CBF {
      then cos-next-hop-map my-cbf-map;
    }
  }
  class-of-service {
    forwarding-policy {
      next-hop-map my-cbf-map {
        forwarding-class best-effort {
          next-hop [ 2001:db8:0:3::3 10.10.10.3 ];
        }
        forwarding-class assured-forwarding {
          next-hop [ 2001:db8:0:5::4 10.10.10.4 ];
        }
      }
    }
  }
}

```

Well, there you have it: CoS for IPv6. Thanks to the straightforward Junos implementation, not a lot has changed from IPv4.

MORE? For more on configuring CoS in IPv6, see the *Junos Software Class of Service Configuration Guide*, available at www.juniper.net/techpubs.

Try It Yourself: Configure CoS in IPv6

You knew it was coming, and you won't be disappointed. It's time for you to take everything you have learned about configuring CoS for IPv6 into your own test bed network and give it a try! Start with building some classifiers and rewrite rules using the `dscp-ipv6` statement. After you apply them to various interfaces, try configuring CBF. Then move on to some more complete CoS configurations. Can you get CoS to work for IPv4 and IPv6 on the same interfaces?

Verifying CoS in IPv6

As mentioned earlier, `dscp-ipv6` is both a configuration statement and a command statement for working with DiffServ in IPv6 packets. Just as in the configuration, you must leverage this command statement in order to verify your IPv6 classifiers and rewrite rules. Let's take a look!

To Verify CoS Policy in IPv6:

1. First verify the classifiers:

```

ipv6@r2> show class-of-service classifier type dscp-ipv6
Classifier: dscp-ipv6-default, Code point type: dscp-ipv6, Index: 8
  Code point      Forwarding class      Loss priority

```

```

000000      best-effort      low
000001      best-effort      low
000010      best-effort      low
000011      best-effort      low
000100      best-effort      low
000101      best-effort      low
000110      best-effort      low
000111      best-effort      low
001000      best-effort      low
001001      best-effort      low
001010      assured-forwarding  low
<snip>

```

Classifier: dscp-ipv6-compatibility, Code point type: dscp-ipv6, Index: 9

```

Code point      Forwarding class      Loss priority
000000      best-effort      low
000001      best-effort      low
000010      best-effort      low
000011      best-effort      low
000100      best-effort      low
000101      best-effort      low
000110      best-effort      low
000111      best-effort      low
001000      best-effort      low
001001      best-effort      low
001010      best-effort      low
<snip>

```

Classifier: my-dscp-map, Code point type: dscp-ipv6, Index: 109

```

Code point      Forwarding class      Loss priority
000000      best-effort      low
001010      assured-forwarding  low
010100      assured-forwarding  low
110000      network-control     low

```

ipv6@r2>

2. Then verify the rewrite rules:

```

ipv6@r2> show class-of-service rewrite-rule type dscp-ipv6

```

Rewrite rule: dscp-ipv6-default, Code point type: dscp-ipv6, Index: 32

```

Forwarding class      Loss priority      Code point
best-effort           low                000000
best-effort           high               000000
expedited-forwarding  low                101110
expedited-forwarding  high               101110
assured-forwarding   low                001010
assured-forwarding   high               001100
network-control       low                110000
network-control       high               111000

```

```

Rewrite rule: my-dscp-rewrite, Code point type: dscp-ipv6, Index: 21793
  Forwarding class      Loss priority      Code point
  best-effort           low                000000
  assured-forwarding    low                001010
  network-control       low                110000

```

```
ipv6@r2>
```

Great! Both the classifier and rewrite rules configured in R2 are functioning as expected.

Junos provides another great tool for verifying the operation of CoS in your network: The TOS statement. TOS stands for type-of-service and although this statement is not IPv6 specific, it is handy enough to be included here anyway.

You can use the TOS statement with both the ping and traceroute commands. In both cases, it is used to specify an IP type-of-service value to be applied to the verification packets sent. This is most useful for verifying the delay and loss of specific traffic classes when used with the ping command and for verifying CBF when used with traceroute.

To Verify CBF with Traceroute:

1. First, traceroute to R1's loopback using the NC class:

```

ipv6@r2> traceroute 2001:db8::1 tos 48
traceroute6 to 2001:db8::1 (2001:db8::1) from 2001:db8:0:1::2, 64 hops max, 12 byte
packets
 1 2001:db8::1 (2001:db8::1) 887.378 ms 6.476 ms 4.086 ms

```

```
ipv6@r2>
```

2. Next, try the same traceroute but use the BE class:

```

ipv6@r2> traceroute 2001:db8::1 tos 0
traceroute6 to 2001:db8::1 (2001:db8::1) from 2001:db8:0:1::2, 64 hops max, 12 byte
packets
 1 2001:db8:0:3::3 (2001:db8:0:3::3) 1.501 ms 2.478 ms 1.427 ms
 2 2001:db8::1 (2001:db8::1) 3.922 ms 4.812 ms 3.755 ms

```

```
ipv6@r2>
```

3. Finally, traceroute again to R1's loopback, this time using the AF class:

```

ipv6@r2> traceroute 2001:db8::1 tos 10
traceroute6 to 2001:db8::1 (2001:db8::1) from 2001:db8:0:1::2, 64 hops max, 12 byte
packets
 1 2001:db8:0:5::4 (2001:db8:0:5::4) 0.072 ms 0.642 ms 0.770 ms

```

```
2 2001:db8::1 (2001:db8::1)      1.922 ms  1.812 ms  1.755 ms
ipv6@r2>
```

You can clearly see that BE traffic uses a next-hop of R3, AF traffic is forwarded to R4, and NC traffic, without CBF applied, takes the IGP's shortest path (direct to R1). And CBF verified!

MORE? To learn more about configuring and verifying CoS in IPv6, check out *Junos Class of Service Using IPv6 DiffServ Feature Guide*, available at www.juniper.net/techpubs.

Try It Yourself: Verify CoS in IPv6

Go apply these lessons in your own test-bed network! Use the `dscp-ipv6` statement to verify your IPv6 CoS policies and then play with the `TOS` statement to ping and traceroute to your heart's content!

Introducing Multicast Listener Discovery

The Multicast Listener Discovery (MLD) protocol is used by IPv6 multicast routers to discover the presence of multicast listeners on directly attached links, and to learn which multicast addresses are of interest to those listeners. Once a router has the information, it passes it on to whichever multicast routing protocol it is using. PIM is a common choice for this process, and is no different from PIM for IPv4 except that IPv6 PIM messages are sent using link-local addresses.

NOTE While MLD performs a function very similar to IGMPv2 in IPv4, it is important to note that MLD uses ICMPv6 messages, and that there is no IGMP for IPv6.

If there is more than one MLD router on a subnet, the routers exchange query messages and the router with the lowest source address becomes the querier for that subnet. All other MLD routers on that subnet become nonqueriers.

The MLD querier router sends MLD queries and receives MLD report messages from any interested listeners. Nonquerier routers are essentially backup multicast routers and can take over as querier should the querier router go down.

MORE? To learn more about Multicast Listener Discovery (MLD), read *RFC 3810 (Multicast Listener Discovery Version 2 (MLDv2) for IPv6)* at <http://www.rfc-editor.org/rfc/rfc3810.txt>. You may also want to review *RFC 2710 (Multicast Listener Discovery (MLD) for IPv6)* and *RFC 3590 (Source Address Selection for the Multicast Listener Discovery (MLD) Protocol)* both of which can be found at <http://www.rfc-editor.org/rfc/>.

NOTE Junos supports both MLDv1 and MLDv2. MLDv2 supports source-specific multicast (SSM).

The best part of the Junos implementation of MLD is that it is enabled automatically on all broadcast interfaces when either PIM or DVMRP is configured. *You read that right; there is no need to configure MLD, unless you want to change the default settings.*

MORE? For more on the Juniper implementation of MLD and how to configure MLD in Junos see Chapters 8, 9, and 10 in *Junos Software Multicast Protocols Configuration Guide*, available at <http://www.juniper.net/techpubs>.

Chapter 3

Discovering IPv6 Enabled System Management

<i>Configuring an IPv6 Backup Router.....</i>	<i>52</i>
<i>Rate Limiting ICMPv6.....</i>	<i>56</i>
<i>IPv6 Path MTU Discovery.....</i>	<i>61</i>
<i>Accepting IPv6 Packets with Zero Hop Limit.....</i>	<i>64</i>
<i>Controlling IPv6 Duplicate Address Detection.....</i>	<i>65</i>
<i>What to Do Next & Where to Go... ..</i>	<i>68</i>

As many experienced network operators come to find out, there is more to running a network than just routing protocols, and this chapter walks you through some of the less glamorous but often essential aspects of network system management.

In Junos, most system management is not IP version specific, so we'll investigate only those system management tools that are IPv6 specific. You will learn how to: configure a backup router using IPv6, rate limit ICMPv6 messages, configure IPv6 path MTU discovery, accept IPv6 packets with a hop-limit of zero, and how to control IPv6 duplicate address detection attempts.

While most of these tools are fairly simple to configure, they can all be lifesavers in certain situations, so read on and add some tricks to your bag of Junos IPv6 configuration tools!

Configuring an IPv6 Backup Router

While a router is booting, the routing protocol process (RPD) is not running. This means that the router has no routes available to it (not even static or default routes), leaving the router unreachable over the network. While a direct console connection or some form of out of band management can often be used to mitigate this potential problem, a backup router is often preferable. You can configure another router as a backup router, to allow the router to boot, and ensure that the router is reachable over the network if the routing protocol process fails to start properly.

NOTE Once RPD starts, the backup router's address is removed from the local routing and forwarding tables.

Although this functionality is not new with IPv6, the command to enable it is different from IPv4. Let's take a look by going back to our book's test bed (as illustrated in Figure 1.1, if you need to refresh yourself).

How To Configure an IPv6 Backup Router on R1

1. First, jump to the `inet6-backup-router` config level, under system:

```
[edit]  
ipv6@r1# edit system inet6-backup-router
```

2. Then add the IPv6 address of the directly connected router which will serve as R1's backup router:

```
[edit system inet6-backup-router]
ipv6@r1# set 2001:db8:0:1::2
```

3. Finish by jumping to the top of the config, verifying your changes, and committing them:

```
[edit system inet6-backup-router]
ipv6@r1# top

[edit]
ipv6@r1# show | compare
[edit system]
+ inet6-backup-router 2001:db8:0:1::2;

[edit]
ipv6@r1# commit
commit complete
```

Wham, bam, thank you Junos! R1 will now use directly connected R2 as its backup router.

NOTE By default all hosts (a default route) are reachable through the backup router.

It is often unacceptable in a production environment to use a default route, even (or perhaps especially) on a temporary basis while the router is booting. For this reason, Junos allows you to configure a specific destination to be reachable through a backup router. In this way you can achieve network reachability while loading, configuring, and recovering a router without the risk of installing a default route.

To Configure a Specific Destination for R1's IPv6 Backup Router:

1. Start by rolling back the previous configuration (to start fresh again):

```
[edit]
ipv6@r1# rollback 1
load complete

[edit]
ipv6@r1# show | compare
[edit system]
- inet6-backup-router 2001:db8:0:1::2;

[edit]
ipv6@r1# commit
commit complete
```

2. Now, jump back to the inet6-backup-router configuration level:

```
[edit]
ipv6@r1# edit system inet6-backup-router
```

3. Next, add the backup router address again, but this time use the destination keyword and add the network to be reachable:

```
[edit system inet6-backup-router]
ipv6@r1# set 2001:db8:0:1::2 destination 2001:db8:0:9::/64
```

4. Then verify and commit the changes:

```
[edit system inet6-backup-router]
ipv6@r1# top

[edit]
ipv6@r1# show | compare
[edit system]
+ inet6-backup-router 2001:db8:0:1::2 destination 2001:db8:0:9::/64;

[edit]
ipv6@r1# commit
commit complete
```

Fantastic! R1 will now use R2 as a backup router with a single reachable prefix.

BEST PRACTICE It's always best to specify the destination and avoid the risks associated with using a default route.

Now that's done, one question remains: "What if I want the route to the backup router to remain, after RPD starts?"

Junos has an answer! To keep the backup router's address in the local routing and forwarding tables when the router is fully operational, just add a static route using the retain keyword.

To Retain a Route to an IPv6 Backup Router:

1. First get into the inet6.0 rib configuration hierarchy:

```
[edit]
ipv6@r1# edit routing-options rib inet6.0
```

2. Then build a static route for the destination, using the backup router's address as the next hop:

```
[edit routing-options rib inet6.0]
```

```
ipv6@r1# set static route 2001:db8:0:9::/64 next-hop 2001:db8:0:1::2
```

3. Next add the retain configuration command:

```
[edit routing-options rib inet6.0]
ipv6@r1# set static route 2001:db8:0:9::/64 retain
```

4. Now all that's left is to verify and commit your changes:

```
[edit routing-options rib inet6.0]
ipv6@r1# top

[edit]
ipv6@r1# show | compare
[edit]
+ routing-options {
+   rib inet6.0 {
+     static {
+       route 2001:db8:0:9::/64 {
+         next-hop 2001:db8:0:1::2;
+         retain;
+       }
+     }
+   }
+ }

c[edit]
ipv6@r1# commit
commit complete
```

There you have it. R1 will direct traffic destined for 2001:db8:0:9::/64 to R2 both before and after RPD has started.

MORE? To learn more about IPv6 backup routers, see *Junos Software System Basics Configuration Guide* found at www.juniper.net/techpubs.

Try It Yourself: Configuring an IPv6 Backup Router

It's your turn again! Jump into your own test bed network and configure some backup routers. Use both the default and explicitly configured destinations. Remember that the backup router must be directly connected (on the same IPv6 subnet). Try using static routes using the **retain** command to force routes to remain in the routing and forwarding tables. Can you configure multiple IPv6 backup routers? How about a combination of IPv4 and IPv6 backup routers? Try using different routers for IPv4 and IPv6.

Rate Limiting ICMPv6

Internet Control Messaging Protocol version 6 (ICMPv6) is based on ICMP for IPv4 but includes several changes. ICMPv6 is an integral part of IPv6, providing both error and informational messages that enable ping, neighbor discovery, router discovery, and Path MTU discovery to name just a few.

While ICMP for IPv6 (ICMPv6) is crucial to the operation of any IPv6 network, it can also be taken advantage of by bad actors in the form of Denial Of Service (DOS) or other attacks. To help combat this, Junos gives us the **icmpv6-rate-limit** configuration command that is used to limit the rate of ICMPv6 messages that are sent.

This command has two required options:

- **Bucket-size**: The number of seconds in the rate-limiting bucket. Acceptable values are 0 through 4294967295 seconds.
- **Packet-rate**: The rate-limiting packets earned per second. Acceptable values are 0 through 4294967295 pps.

To Enable ICMPv6 Rate Limiting on R2:

1. First jump to the system internet-options configuration hierarchy:

```
[edit]
ipv6@r2# edit system internet-options
```

2. Then use the `icmpv6-rate-limit` command to set both the bucket-size and the packet-rate to the desired values:

```
[edit system internet-options]
ipv6@r2# set icmpv6-rate-limit bucket-size 5 packet-rate 1000
```

3. Finally, verify and commit your changes:

```
[edit system internet-options]
ipv6@r2# top
```

```
[edit]
ipv6@r2# show | compare
[edit system]
+ internet-options {
+   icmpv6-rate-limit packet-rate 1000 bucket-size 5;
+ }
```

```
[edit]
ipv6@r2# commit
commit complete
```

MORE? To learn more about ICMP for IPv6 (ICMPv6), check out *RFC 4443 (Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification)* at tools.ietf.org/rfc/rfc4443.txt.

Using Policers and Firewall Filters

In addition to the **icmpv6-rate-limit** command, you can also leverage policers and firewall filters to help control the rate of ICMPv6 messages in your network.

There are two primary differences between configuring IPv4 ICMP filters and IPv6 ICMPv6 filters:

- **family inet6:** All IPv6 firewall filters are built under family inet6 rather than IPv4's family inet.
- **next-header:** You may be accustomed to using the protocol configuration stanza when building IPv4 filters, for IPv6 this is replaced by the term next-header.

To see these changes in action, let's walk through the following configuration example.

To Configure ICMPv6 Policing on R2:

1. Start by getting into the `firewall` configuration:

```
[edit]
ipv6@r2# edit firewall
```

2. Then create your policer:

```
[edit firewall]
ipv6@r2# edit policer ICMPv6_20m
```

3. Now configure the policer with a 20 Mbps rate limit and a 625 KB maximum burst size:

```
[edit firewall policer ICMPv6_20m]
ipv6@r2# set if-exceeding bandwidth-limit 20m
```

```
[edit firewall policer ICMPv6_20m]
ipv6@r2# set if-exceeding burst-size-limit 625k
```

```
[edit firewall policer ICMPv6_20m]
ipv6@r2# set then discard
```

4. Next, create your firewall filter and first term:

```
[edit firewall policer ICMPv6_20m]
ipv6@r2# up
```

```
[edit firewall]
ipv6@r2# edit family inet6 filter POLICE term ICMPv6
```

5. Now configure this term to match ICMPv6 packets and police them using the policer you just built:

```
[edit firewall family inet6 filter POLICE term ICMPv6]
ipv6@r2# set from next-header icmpv6
```

```
[edit firewall family inet6 filter POLICE term ICMPv6]
ipv6@r2# set then policer ICMPv6_20m
```

6. Next, apply the filter to R2's LAN facing interface:

```
[edit firewall family inet6 filter POLICE term ICMPv6]
ipv6@r2# top edit interfaces ge-1/0/1 unit 100 family inet6
```

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r2# set filter input POLICE
```

7. Then verify your changes and commit:

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r2# top
```

```
[edit]
ipv6@r2# show | compare
[edit interfaces ge-1/0/1 unit 100 family inet6]
+   filter {
+     input POLICE;
+   }
[edit]
+ firewall {
+   family inet6 {
+     filter POLICE {
+       term ICMPv6 {
+         from {
+           next-header icmpv6;
+         }
+         then policer ICMPv6_20m;
+       }
+     }
+   }
+   policer ICMPv6_20m {
+     if-exceeding {
+       bandwidth-limit 20m;
+       burst-size-limit 625k;
+     }
+   }
+ }
```

```
+         then discard;
+     }
+ }
```

```
[edit]
ipv6@r2# commit
commit complete
```

There you have it; R2 will not accept any more than 20 Mbps of ICMPv6 traffic into interface ge-1/0/1.100 now.

TIP You can also configure a policer to rate limit traffic by percentage of interface bandwidth.

BEST PRACTICE To defend against attacks, rate limit all ICMPv6 and Hop-by-Hop (HbH) options.

Going a step further, you may want to lock ICMPv6 traffic down a bit tighter by only allowing messages required for neighbor Discovery, Router Advertisements and PMTU discovery. Junos allows you granular filter control within family inet6 to do just that. After adding the ICMPv6_20m policer to R1, let's explore this functionality.

To Configure ICMPv6 Policing and Filtering on R1:

1. Start by creating the IPv6 filter and first term:

```
[edit]
ipv6@r1# edit firewall family inet6 filter ICMPv6 term POLICE
```

2. Next, configure this term to match only essential ICMPv6 message types:

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set from icmp-type packet-too-big
```

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set from icmp-type router-advertisement
```

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set from icmp-type router-solicit
```

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set from icmp-type neighbor-advertisement
```

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set from icmp-type neighbor-solicit
```

3. Now set this term to apply the 20M policer to matched traffic:

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# set then policer ICMPv6_20m
```

4. Then create your next firewall filter term:

```
[edit firewall family inet6 filter ICMPv6 term POLICE]
ipv6@r1# up
```

```
[edit firewall family inet6 filter ICMPv6]
ipv6@r1# edit term DROP
```

5. Now configure this term to match all remaining ICMPv6 packets and discard them:

```
[edit firewall family inet6 filter STOP term DROP]
ipv6@r1# set from next-header icmpv6
```

```
[edit firewall family inet6 filter STOP term DROP]
ipv6@r1# set then discard
```

6. Next, apply the filter to R1's LAN facing interface:

```
[edit firewall family inet6 filter ICMPv6 term DROP]
ipv6@r1# top edit interfaces ge-1/0/1 unit 100 family inet6
```

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r1# set filter input ICMPv6
```

7. Finally, verify the changes and commit:

```
[edit interfaces ge-1/0/1 unit 100 family inet6]
ipv6@r1# top
```

```
[edit]
ipv6@r1# show | compare
[edit interfaces ge-1/0/1 unit 100 family inet6]
+   filter {
+     input POLICE;
+   }
[edit firewall family inet6]
+   filter ICMPv6 {
+     term POLICE {
+       from {
+         icmp-type [ packet-too-big router-advertisement router-solicit
neighbor-advertisement neighbor-solicit ];
+       }
+       then policer ICMPv6_20m;
+     }
+     term DROP {
+       from {
+         next-header icmpv6;
```

```
+         }  
+         then discard;  
+     }  
+ }
```

```
[edit]  
ipv6@r1# commit  
commit complete
```

Now R1 will only accept up to 20 Mbps of critical ICMPv6 traffic into interface ge-1/0/1.100, the rest will be dropped.

MORE? For more on Junos firewall filters and policers, check out Chapters 8 through 12 in *Junos Software Policy Framework Configuration Guide*, available at www.juniper.net/techpubs.

Try It Yourself: Rate Limiting ICMPv6

Now you can take this into your own lab network and try it out. Start with the `icmpv6-rate-limit` configuration command to limit the rate that ICMPv6 messages are sent. Try different packet-rate and bucket-size settings; how do they affect your network?

Next try getting more creative with policers and firewall filters. There are lots of options to test here. Start trying both `bandwidth-limit` and `bandwidth-percent` to set the rate. Can you set the limit low enough to effect network operation? Experiment with different `burst-size-limit` settings as well. Move on to applying the policer(s) to various protocols using the `next-header` option. Try allowing and denying specific ICMPv6 message types. What other types of IPv6 filters can you build?

Tell the other readers of this Day One book what you found or experimented on. Post your results, issues, and questions on this book's pages on J-Net. Go to www.juniper.net/dayone.

IPv6 Path MTU Discovery

Path MTU discovery is a method used to determine the largest packet size that can travel between two nodes without being fragmented. The size of that packet is considered the Path Maximum Transmission Unit (PMTU) and is equal to the smallest link MTU of all the links along that path. Most IPv6 hosts implement PMTU discovery in order to use the largest packet size possible over a given path which allows optimal throughput.

PMTUD works by sending the largest packet allowed by the local MTU to the far end of the path. If the packet makes it through, the local MTU is used as the PMTU. If, however, any node along the path has a

lower MTU (and thus can not forward the packet), that node will discard the packet and return an ICMPv6 packet too big message. The process repeats until the sending node gets a packet through (finds the PMTU) or fails back to the IPv6 minimum MTU.

NOTE Because IPv6 only allows packet fragmentation at the source node, PMTU discovery is used by almost all IPv6 hosts – the alternative is to use the IPv6 minimum link MTU for all packets (1280 bytes).

Since Junos enables PMTU discovery by default, in many cases you will not need to pay it a second thought. There are, however, situations that may require you to disable PMTU discovery or to change the default timeout to better fit your network topology.

The most common reason to disable PMTUD is when all ICMPv6 messages are being blocked somewhere along the path. If a sender does not receive ‘packet too big’ messages for any reason, that sender has no way to know that its packets were dropped and will incorrectly assume that their local MTU is the correct PMTU. This results in data packets larger than the actual PMTU being blackholed by the node with a smaller MTU as it discards the larger packets but has no way to notify the sender.

Routing topologies often change over time, and because of this PMTU discovery periodically raises the assumed PMTU for each path in order to rediscover the optimum PMTU. How often this check is performed is determined by the PMTUD timeout. You may want to change the PMTUD timeout if your routing topology changes frequently, is very stable, or if you are sure that it will not change at all.

MORE? If you want to know more about Path MTU discovery read RFC 1981 *Path MTU Discovery for IP version 6*. Find it at: tools.ietf.org/rfc/rfc1981.txt.

Let’s now take a look at how to disable PMTU discovery and how to change the default PMTUD timeout in Junos.

To Disable Path MTU Discovery on R1:

1. First move into the internet-options configuration level, under system:

```
[edit]  
ipv6@r1# edit system internet-options
```

2. Now set the `no-ipv6-path-mtu-discovery` configuration command:

```
[edit system internet-options]
ipv6@r1# set no-ipv6-path-mtu-discovery
```

3. Then jump to the top of the config, verify your changes, and commit:

```
[edit system internet-options]
ipv6@r1# top

[edit]
ipv6@r1# show | compare
[edit system]
+ internet-options {
+   no-ipv6-path-mtu-discovery;
+ }
```

```
[edit]
ipv6@r1# commit
commit complete
```

There you have it, R1 will no longer use Path MTU discovery. After rolling that change back, it's time to explicitly configure the PMTU discovery timeout.

To Configure the Path MTU Discovery Timeout on R1:

1. Start by jumping back to the system, internet-options configuration level:

```
[edit]
ipv6@r1# edit system internet-options
```

2. Now configure the PMTU discovery timeout in minutes:

```
[edit system internet-options]
ipv6@r1# set ipv6-path-mtu-discovery-timeout 4
```

3. As always, the last step is to verify your changes and commit:

```
[edit system internet-options]
ipv6@r1# top

[edit]
ipv6@r1# show | compare
[edit system]
+ internet-options {
+   ipv6-path-mtu-discovery-timeout 4;
+ }
```

```
[edit]
ipv6@r1# commit
commit complete
```

Test bed router R1 now has a PMTU discovery timeout of 4 minutes.

NOTE The default Path MTU discovery timeout in Junos is 10 minutes.

MORE? To learn more about configuring PMTU discovery, see *Junos Software System Basics Configuration Guide*, at www.juniper.net/techpubs.

Try It Yourself: Configuring IPv6 Path MTU Discovery

Take what you have learned here to your own lab or test-bed network. Try configuring the PMTU discovery timeout and disabling PMTU discovery altogether. What happens if you disable PMTU discovery on only one node in your network? Use the ping command with the size option to find the PMTU for various paths in your test bed.

Accepting IPv6 Packets with Zero Hop Limit

The IPv6 Hop Limit is an 8-bit integer which is decremented by 1 at each node that the packet transits. In other words, it provides a very similar function to, and is basically a rename of, IPv4's Time To Live (TTL) field.

A packet whose hop limit is decremented to zero should be discarded to prevent forwarding loops. For this reason, the default behavior of Junos rejects incoming packets with a hop limit of zero. You may, however, find yourself in a situation where you need a Junos device to accept packets addressed to the local host even when they have a hop limit of zero. Perhaps another vendor's network device is misbehaving or an application is incorrectly marking hop limits on packets destined for this router. To address this, and to allow you to control this behavior, Junos provides the `no-ipv6-reject-zero-hop-limit` and `ipv6-reject-zero-hop-limit` configuration statements.

To Accept IPv6 Packets with a Hop Limit of Zero on R2:

1. First, get into the system, internet-options configuration hierarchy level:

```
[edit]
ipv6@r2# edit system internet-options
```

2. Then apply the `no-ipv6-reject-zero-hop-limit` statement:

```
[edit system internet-options]
ipv6@r2# set no-ipv6-reject-zero-hop-limit
```

3. Now verify and commit your change:

```
[edit system internet-options]
ipv6@r2# top

[edit]
ipv6@r2# show | compare
[edit system internet-options]
+ no-ipv6-reject-zero-hop-limit;

[edit]
ipv6@r2# commit
commit complete
```

R2 will now accept packets with a zero hop limit that are addressed to it; transit packets will still be rejected.

In production, accepting IPv6 packets with a hop limit of zero will be very rare and should only be used as a temporary fix while the hop limit in the offending packets is fixed, if at all possible.

NOTE You can use the complimentary `ipv6-reject-zero-hop-limit` statement to restore the default behavior of rejecting all packets with a hop limit of zero.

MORE? For a bit more on Hop Limit in IPv6, see RFC 2460 *Internet Protocol, Version 6 (IPv6) Specification*, found at <http://tools.ietf.org/rfc/rfc2460.txt>.

Try It Yourself: Accepting IPv6 Packets with Zero Hop Limit

This is an easy Try It Yourself. First set up your own test bed network to send packets with a zero hop limit to one of your Junos devices. Then use the `no-ipv6-reject-zero-hop-limit` and `ipv6-reject-zero-hop-limit` configuration statements to accept and deny those packets. Try sending transit packets with a zero hop limit in both configurations as well.

Controlling IPv6 Duplicate Address Detection

The last tool that you will add to your IPv6 network operation tool box in this section is the `ipv6-duplicate-addr-detection-transmits` statement. This configuration command allows you to control the number of attempts a Junos device makes for duplicate address detection.

You may need to take advantage of this capability if your router is on an exceptionally lossy network where many duplicate address detection messages might be lost (of course this will cause plenty of more pressing

issues as well) or, more likely, in situations where many nodes will all come online at the same time. It's also possible that you may need to speed up the address assignment process by lowering the number of duplicate address detection attempts, in situations where collisions are rare or not expected.

To Set the Number of Duplicate Address Detection Attempts on R2:

1. Start by jumping to the system, internet-options configuration hierarchy level:

```
[edit]
ipv6@r2# edit system internet-options
```

2. Then use the `ipv6-duplicate-addr-detection-transmits` command to set the number of attempts to 9:

```
[edit system internet-options]
ipv6@r2# set ipv6-duplicate-addr-detection-transmits 9
```

3. Last, verify your change and then commit it:

```
[edit system internet-options]
ipv6@r2# top
```

```
[edit]
ipv6@r2# show | compare
[edit system internet-options]
+ ipv6-duplicate-addr-detection-transmits 9;
```

```
[edit]
ipv6@r2# commit
commit complete
```

This book's test bed router R2 will now make nine attempts for IPv6 duplicate address detection before assigning a unicast address to an interface. That's a bit much, but this is just a testbed, right?

NOTE The Junos default is three duplicate address detection attempts.

MORE? For more information about duplicate address detection, take a look at RFC 4429 *Optimistic Duplicate Address Detection (DAD) for IPv6*, RFC 2462 *IPv6 Stateless Address Autoconfiguration*, and RFC 2461 *Neighbor Discovery for IP Version 6 (IPv6)*, all found at <http://tools.ietf.org/rfc/>.

What to Do Next & Where to Go ...

<http://www.juniper.net/dayone>

The PDF format of this book includes an additional Appendix.

<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 at this free forum.

<http://www.juniper.net/techpubs/software/junos>

The Junos technical documentation includes everything you need to understand and configure all aspects of Junos, including IPv6.

<http://www.ipv6forum.com>

The IPv6 Forum is a world-wide consortium of Internet service vendors, National Research & Education Networks (NRENs), and international ISPs whose focus is to provide technical guidance for the deployment of IPv6.

<http://www.getipv6.info>

The American Registry for Internet Numbers (ARIN) hosts this IPv6 Wiki which provides a wide variety of information on IPv6.

<http://www.juniper.net/us/en/products-services/technical-services/j-care/>

Building on the Junos automation toolset, Juniper Networks Advanced Insight Solutions (AIS) introduces intelligent self-analysis capabilities directly into platforms run by Junos. AIS provides a comprehensive set of tools and technologies designed to enable Juniper Networks Technical Services with the automated delivery of tailored, proactive network intelligence and support services.

<http://www.theipv6experts.net>

The IPv6 Experts.net is a coalition of recognized industry experts in all aspects of IPv6 who provide insight and advice on many aspects of IPv6.