JUNIPER NETWORKS

ACCESS & AGGREGATION | EDGE | CORE | DATA CENTER | WAN | CAMPUS & BRANCH | CONSUMER & BUSINESS DEVICE

Junos® Fundamentals

DAY ONE: JUNOS FOR IOS ENGINEERS

When it's time to swap out your Cisco routers with new Juniper Networks devices, use this book to expand your skill set to include Junos, and make the transition to an advanced operating system with speed and confidence.

By Chris Jones

DAY ONE: JUNOS FOR IOS ENGINEERS

Book #8 in the Junos Fundamentals series

There comes a time in every network engineer's career when they need to transition to other technologies, but appearances of complexity can get in the way. To make the path from IOS to Junos easier, you need a book that can make you feel confident about how your new Juniper Network router, switch, or security device operates.

Day One: Junos for IOS Engineers addresses the needs of the IOS-trained engineer by providing a side-by-side comparison of configurations and techniques in both IOS and Junos. In a few quick steps you can compare what you did yesterday with IOS to what you can do today with Junos. Along the way are insights, tips, and no-nonsense explanations of what is taking place. If you are an engineer who is already familiar with IOS, get ready to see the 'Junos way' in action, whether it's simply using a different syntax or a whole new efficient way of networking.

"This Day One book is just what was needed for those of us who know IOS but need to know Junos. The examples being shown side-by-side between IOS and Junos really helps us understand the Junos configuration."

Jeff Fry, CCIE #22061

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

- Understand the differences in thought process between Junos and IOS.
- Configure simple functions and common tasks on any Junos device.
- Compare Junos configurations to your accustomed IOS configurations.
- Write configurations for small networks utilizing VLANs, OSPF, and BGP.

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

Published by Juniper Networks Books

ISBN 978-1936779543

51600

9 781936 779543

7100157

JUNIPER NETWORKS

Junos® Fundamentals

Day One: Junos for IOS Engineers

By Chris Jones

JUNIPER NETWORKS

Published by Juniper Networks Books
Author: Chris Jones
Key Contributor: Dana Yanch
Technical Reviewers: Tom Dwyer, Nick Ryce, and Stefan Fouant
Editor in Chief: Patrick Ames
Copyeditor and Proofer: Nancy Koerbel
J-Net Community Manager: Julie Wider

About the Author
Chris Jones is a Network Engineer, certified with Juniper as JNCIE-ENT #272, and with Cisco Systems as CCIE #25655 (R&S). Chris has more than eight years of industry experience with both Cisco and Juniper products and solutions.

Author's Acknowledgments
I would like to thank my wife Jennifer, for all the support she has given me during the long hours spent writing this book, not to mention her patience in me while studying for my CCIE and my JNCIE. Special thanks goes out to my good friend Dana Yanch, who provided the IOS configurations for this book. I couldn't have done it without him! Last but not least, I would also like to thank the Editor, Patrick Ames, for giving me the opportunity to write this Day One guide. His guidance has been crucial to the successful completion of the book.

ISBN: 978-1-936779-54-3 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-936779-55-0 (ebook)

Version History: v1 August 2012
2 3 4 5 6 7 8 9 10 #7100157-en

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

About the Contributors and Technical Reviewers

Books such as this are seldom objective and one could argue that to be truly useful, they can't be. However, the following contributors helped to make this book as straightforward and as objective as possible.

- **Dana Yanch**, CCIE #25567 and JNCIP-ENT, is a Network Architect with Insight Enterprises. Dana was a prime contributer to the writing of this book, providing all of the Cisco IOS configurations as an independent source.
- **Tom Dwyer** is a Senior Network Security Engineer for Nexum Inc. He has 18 years of networking experience and holds several Juniper certifications including JNCIP-ENT/SEC and JNCI.
- **Nick Ryce** is a Senior Network Engineer with Pulsant in the UK. He is certified with Juniper as JNCIE-ENT #232 and has worked within the IT industry for over seven years. Nick contibuted his skills to this book in the form of a technical review.
- **Stefan Fouant** is a Technical Trainer and JNCP Proctor at Juniper Networks. He is one of a few triple-JNCIEs and this book is indebted to his technical review of both OSes.

Comparing IOS to Junos

After many discussions, the editor, the contributors, and the author decided to present the two OSes as simple comparisons, one after the other, with just a smattering of point-by-point comparisons. The reader is expected to read through the Cisco configurations as something they should already be familiar with, and then read through the corresponding Junos configurations where notations and distinctions are offered.

This book begins with a chapter on tasks common to the two operating systems, proceeds with a chapter on comparing common configurations between the two, and concludes with a final chapter comparing a configuration of a simple but total network.

This is a book to help IOS Engineers administer Juniper devices in a single day.

What You Need to Know Before Reading This Book

- A fundamental knowledge of the Junos operating system is necessary to accomplish the tasks in this book. While this book aims to compare common configurations between IOS and Junos, it does not cover things such as operating within the Junos CLI.
- This book assumes you have reviewed and read two *Day One* books on Junos – *Day One: Exploring the Junos CLI*, and *Day One: Configuring Junos Basics*. Both are available for free download at http://www.juniper.com/dayone.
- This book assumes that the reader is well versed in IOS and is familiar with configuring basic administrative tasks, as well as simple IGP and BGP scenarios.
- Finally, this book assumes an understanding of general network protocols, such as OSPF and BGP.

After Reading This Book, You'll be Able To:

- Understand the differences in thought processes between Junos and IOS.
- Configure simple functions and common tasks on any Junos router.
- Compare Junos configurations to IOS configurations that you are accustomed to.
- Write configurations for small networks utilizing VLANs, OSPF, and BGP.

Welcome to Day One

Day One books help you to quickly get started in a new topic with just the information that you need on day one. The *Day One* series covers the essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow, while also providing lots of references on where to learn more. For more information on the complete *Day One* library, see http://www.juniper.net/dayone.

Chapter 1

Fundamentals

This book will help IOS Engineers compare common networking tasks to their Junos equivalents. The tasks may seem too basic at first, but as with any new language, learn the small, common daily things first.

MORE? If you missed it in the front matter, the author assumes you know Junos basics. You should be familiar with these books in the *Day One* library: *Exploring the Junos CLI* and *Configuring Junos Basics*. Both are available for free download at http://www.juniper.com/dayone, or are available at the iTunes iBookstore and at Amazon's Kindle bookstore.

This chapter covers basic tasks that an engineer will likely be required to configure on a Juniper router:

- Setting a Hostname
- Creating a User
- Enabling SSHv2
- Assigning an IP Address to an Interface
- Saving a Configuration
- Upgrading the Software

Setting a Hostname

The hostname is one of the most common configurations one can configure on a router, but it is essential for identifying the device during regular configuration as well as when performing logging. Let's compare how each OS does it.

IOS Configuration

To Set the Hostname on an IOS Device

```
Router#configure terminal
Router(config)#hostname R1
R1(config)# end
R1#
```

To Verify the Hostname Has Been Set on an IOS Device

The router prompt reflects the new hostname, but you can also see the configured hostname in the running-config:

```
R1#show running-config | include hostname
hostname R1
R1#
```

Junos Configuration

To Set the Hostname on a Junos Device

```
cjones@router> configure
cjones@router# set system host-name R1
cjones@router# commit and-quit
cjones@R1>
```

To Verify the Hostname Has Been Set on a Junos Device

Again, the router prompt also reflects the new hostname. You can also see the configured hostname in the configuration:

```
cjones@R1> show configuration system | match host-name
host-name R1;
```

Summary

As you can see, the host name configuration for IOS and Junos is very similar, consisting of only a single command. So far, so good.

You can certainly notice the way the two OSes have a different look and feel. By the end of this book, you'll know why this is so and should be comfortable with the change. Let's try another task and pay attention to this different parlance.

Create a User

Creating users is an important part of the configuration. Let's compare allowing multiple engineers to each have access to the device, with varying levels of rights for viewing the configuration and making changes.

IOS Configuration

To Create a User with a Plain-text Password in IOS

```
R1#configure terminal
R1(config)#username plaintextuser password p@$swORd
R1(config)#end
R1#
```

To Create a user with an Encrypted Password in IOS

```
R1#configure terminal
R1(config)#username encrypteduser secret p@$swORd
R1(config)#end
R1#
```

To Verify the Users Have Been Created in IOS

```
R1#show running-config | include username
username plaintextuser password 0 p@$swORd
username encrypteduser secret 5 $1$kbb9$i/TLOSPZr.F8g2LD/MFVi0
R1#
```

Junos Configuration

To Configure a User in Junos

```
cjones@R1> configure
cjones@R1# set system login user bob class super-user full-name "Bob" authentication
plain-text-password
New password:
Retype new password:

cjones@router# commit and-quit
cjones@R1>
```

To Verify User Creation in Junos

```
cjones@R1> show configuration system login
user bob {
    full-name Bob;
    uid 2001;
    class super-user;
    authentication {
```

```
        encrypted-password "$1$wkvA6Ppe$NHN5HgacMAE4OYlvMxF7j/"; ## SECRET-DATA
    }
}

cjones@R1>
```

Summary

User configuration in Junos is slightly more complex than it is in IOS, but is also significantly more flexible right out of the box. The ability to assign pre-defined user roles is a clear advantage.

Notice that passwords in Junos are also hashed by default.

Let's stop here for a moment and discuss the way the Junos configuration displays. This example shows the standard Junos configuration output with curly braces. This may seem intimidating at first (it's *not* C programming), but once you get a handle on how things are laid out, you'll find it far more intuitive than in IOS, where configuration sections can have no rhyme or reason. And once your configurations get large, you'll be thankful for this output format.

Junos Tip In contrast, sometimes it's helpful to be able to see the configuration as it would be typed, in a list of set commands. To do this, just use `| display set` after `show configuration`:

```
cjones@R1> show configuration | display set
set system login user bob full-name Bob
set system login user bob uid 2001
set system login user bob class super-user
set system login user bob authentication encrypted-password "$1$wkvA6Ppe$NHN5HgacMAE4O
YlvMxF7j/"
```

This can also be done in edit mode, omitting the `configuration` keyword:

```
cjones@R1# show | display set
set system login user bob full-name Bob
set system login user bob uid 2001
set system login user bob class super-user
set system login user bob authentication encrypted-password "$1$wkvA6Ppe$NHN5HgacMAE4O
YlvMxF7j/"
```

Enabling SSHv2

SSHv2 is an authentication protocol that uses encrypted communication, unlike Telnet, which is unencrypted. Let's see how the two OSes compare on this basic task.

IOS Configuration

To Configure SSHv2 in IOS

```
R1#configure terminal
R1(config)#ip domain-name juniper.net
R1(config)#crypto key generate rsa modulus 1024
R1(config)#ip ssh version 2
R1(config)#end
R1#
```

To Verify SSHv2 in IOS

```
R1#show ip ssh
SSH Enabled - version 2.0
Authentication timeout: 120 secs; Authentication retries: 3
R1#
```

Junos Configuration

To Configure SSHv2 in Junos

```
cjones@R1> configure
cjones@R1# set system services ssh
cjones@R1# commit and-quit
cjones@R1>
```

To Verify SSHv2 in Junos

```
cjones@R1> show configuration system services
ssh;
```

Summary

Configuring SSHv2 in Junos is a bit more straightforward, since RSA keys are already generated during the OS installation, thanks to Junos's FreeBSD roots. IOS, on the other hand, requires a domain name to be set before it will generate RSA keys.

The default version of SSH in IOS is 1.99, whereas the default version in Junos is Version 2.

NOTE Any firewall filters that may have been applied will be need to be modified in order to accept SSH traffic on TCP/22.

Assign an IP Address to an Interface

Assigning IP addresses is a fundamental task required for any network configuration, as IP addresses are required for devices to be able to communicate. But this fundamental task shows off some major ways in which the two OSes see differently.

IOS Configuration

To Configure an IP Address on an Interface in IOS

```
R1#configure terminal
R1(config)#interface f0/0
R1(config-if)#ip address 192.0.2.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#end
R1#
```

To Verify the IP Address on the Interface in IOS

```
R1#show ip interface brief FastEthernet 0/0
Interface                  IP-Address      OK? Method Status      Protocol
FastEthernet0/0            192.0.2.1       YES manual up          up
```

Junos Configuration

To Configure an IP Address on an Interface in Junos

The syntax in this Junos example specifies a number of things, all in one command:

- The interface name (ge-0/0/0)
- The logical unit number (0, in this case)
- The address family (inet – which is the family used for an IPv4 address)
- The interface address itself (192.0.2.1/24)

NOTE The logical unit number for an interface in Junos is essentially the equivalent of a *sub-interface* in IOS. However, in Junos the unit number is always required in order to enforce consistency. Logical configurations (for example, IP addresses) reside under the unit. Physical configurations (for example, speed, duplex) reside under the physical interface in the hierarchy. The unit number is typically 0 unless a special configuration is present, such as VLAN tagging.

```
cjones@R1> configure
cjones@R1# set interfaces ge-0/0/0 unit 0 family inet address 192.0.2.1/24
```

```
cjones@R1# commit and-quit
cjones@R1>
```

To Verify the IP Address on the Interface in Junos

```
cjones@R1> show configuration interfaces fe-0/0/0
unit 0 {
    family inet {
        address 192.0.2.1/24;
    }
}


cjones@R1> show interfaces terse fe-0/0/0
Interface               Admin Link Proto    Local                 Remote
fe-0/0/0                up    up
fe-0/0/0.0              up    up   inet     192.0.2.1/24
```

Summary

Configuring an IP address on an interface in Junos requires only a single command. Note also that unlike IOS, which requires a subnet mask to be entered in dotted decimal format, the Junos OS allows you to use CIDR notation.

NOTE In Junos, failure to specify a subnet mask will always result in a /32 address being configured.

In IOS, an additional IP address configured on an interface will overwrite an already configured address, unless the `secondary` keyword is added. By comparison, when configuring an additional address on an interface in Junos, the result will be both the original and the new addresses on the interface.

In IOS, an interface is administratively shut down by default, requiring the `no shutdown` command to enable the interface. In Junos, the interfaces are enabled by default and may be shut down with `set interfaces <interface> disable`, and committing.

Saving a Configuration

Saving your configuration is a vital part of managing a router or switch – you will lose your configuration if the device is rebooted. This basic task illustrates the different ways that commits are done by the two OSes.

IOS Configuration

To Save a Configuration in IOS

```
R1#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
R1#
```

To Verify the Configuration Has Been Saved in IOS

Verifying the configuration requires checking the nvram: filesystem for the existence of a startup-config file with a non-zero size:

```
R1#dir nvram:
Directory of nvram:/

   52  -rw-         839                    <no date>  startup-config
   53  ----        1924                    <no date>  private-config
    1  ----          15                    <no date>  persistent-data
    2  -rw-           0                    <no date>  ifIndex-table

57336 bytes total (52473 bytes free)
R1#
```

Junos Configuration

In Junos, the active configuration is the same as the configuration that will be loaded when the device is booted.

Saving a candidate configuration in Junos is done with the `commit` command. Committing will activate your candidate configuration, and notify you if there are any issues with your configuration before the changes are applied.

You can also use the `commit check` command to verify your configuration without actually committing it.

Finally, you can use the `commit confirmed` command to cause the router to roll back to the last good configuration after a specified amount of time, which is especially useful when committing any major changes to the firewall filters or routing policy. To prevent the software from rolling back the configuration, simply type a second `commit`

within the commit confirmed time window, which is 10 minutes by default.

To Save a Configuration in Junos

```
cjones@R1> configure
cjones@R1# commit and-quit
cjones@R1>
```

To Verify the Configuration Has Been Saved in Junos

The best way to verify that a configuration has been saved is to compare it to the previous version. In this case, let's compare rollback number 0 (the current configuration) with rollback number 1 (the previously committed configuration).

```
cjones@R1> show system rollback compare 1
[edit interfaces]
-   fe-0/0/0 {
-       unit 0 {
-           family inet {
-               address 192.0.2.1/24;
-           }
-       }
-   }
```

Summary

In IOS, entered commands take effect immediately. To save your configuration, you must copy your running configuration to your startup configuration. In contrast, Junos uses a *commit model*, which allows you to enter your configuration commands without them having an immediate impact on your device.

Junos also allows you to automatically roll back a configuration with the `confirmed` keyword. IOS requires you to configure the router in order to roll back to the startup-config.

Upgrading the Software

Upgrading software is a vital task that any engineer is required to perform. It is essential that a router or switch be running a code version that balances stability and features. It is typically best to run the latest support-recommended release, unless you require a feature that is only present in a newer version. Let's see how the two OSes compare.

IOS Configuration

To Upgrade the IOS Software

```
R1#copy tftp: flash:

Address or name of remote host []? 192.0.2.2
****192.0.2.2 is the IP address of the TFTP server
Source filename []? c2600-adventerprisek9-mz.124-12.bin
Destination filename [c2600-adventerprisek9-mz.124-12.bin]?y

Accessing tftp://192.0.2.2/c2600-adventerprisek9-mz.124-12.bin...
Erase flash: before copying? [confirm]y

****If there is not enough memory available, erase the Flash.

Erasing the flash filesystem will remove all files! Continue? [confirm]y

Erasing device... eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
...erased

Erase of flash: complete

Loading c2600-adventerprisek9-mz.124-12.bin from 192.0.2.2 (via FastEthernet0/0): !!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!

[OK - 29654656/49807356 bytes]

Verifying checksum...  OK (0xAC8A)

29654656 bytes copied in 56.88 secs (80383 bytes/sec)

R1#reload
Proceed with reload? [confirm]
```

To Verify the IOS Software Has Been Upgraded

```
R1#show version | include IOS
Cisco IOS Software, 2600 Software (C2600-ADVENTERPRISEK9-MZ), Version 12.4(12)T9,
RELEASE SOFTWARE (fc5)
R1#
```

Junos Configuration

To Upgrade the Junos Software

Upgrading the software on a Junos router is very simple. Thanks to its FreeBSD roots, moving the software package to a router is very simple. The first step is to use SCP, FTP, or TFTP to copy the software package to the /var/tmp directory (it can be placed elsewhere, but /var/tmp is the typical location). Then, simply run the following command:

```
cjones@R1> request system software add reboot /var/tmp/ junos-srxsme-10.4R7.5-
domestic.tgz
```

The device will reboot, and the new software will be installed.

To Verify the Junos Software Has Been Upgraded

```
cjones@R1> show version
Hostname: R1
Model: srx100h
JUNOS Software Release [10.4R7.5]
```

Summary

Upgrading both IOS software and Junos software is straightforward, and only requires one command, however, in the case of IOS, a separate reload command is also required.

Conclusion

Are you getting used to the change of appearance between the two OSes? Remember that you can use `| display set` after `show configuration` in Junos to show Junos is similar to something you might be used to. Unfortunately, IOS does not have a reciprical to display Junos hierarchy.

Okay, these were basic concepts that you can now build upon in the coming chapters. You opened the Juniper device, established user accounts, and even upgraded the software. Now, let's do a basic configuration walk-through.

Chapter 2

Basic Configuration

Since this is a *Day One* book, you learn by doing. And this chapter doesn't beat around the bush. It shows you how to build configurations that are usable in a real network. Each example details a specific technology, and a summary at the end of the example draws comparisons and conclusions about the differences between the IOS and Junos configurations.

Using the configurations to study going from IOS to Junos, compare sequences, match commands, and get accustomed to the output.

Single-Area OSPF

Let's build a small three-router network and apply OSPF configurations in order to exchange routing information between routers.

Figure 2.1 illustrates our little setup, a simple three-router topology configured as a single-area OSPF network.

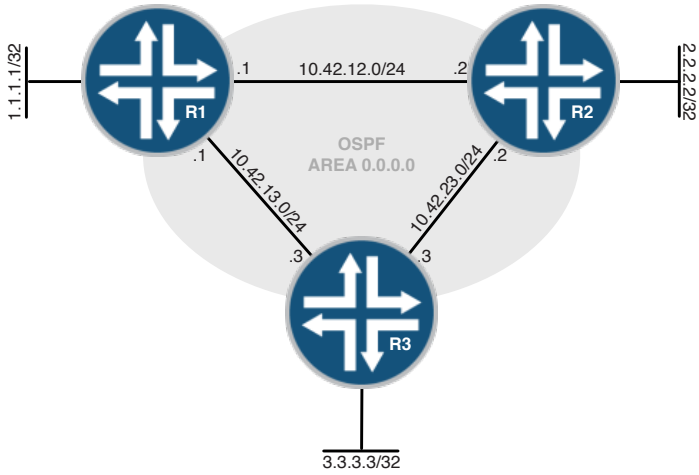

Figure 2.1 Single-Area OSPF Topology

And our task list in this section is to:

- Configure OSPF between R1, R2, and R3
- Advertise loopback interfaces on all routers into OSPF and ensure loopbacks are configured as passive
- Configure the OSPF router-id manually

IOS Configuration

Let's first configure this network using IOS using the following steps.

To configure the IOS routers for initial connectivity

In order for the routers to be able to communicate, IP addressing must be configured on connected interfaces. The following are the steps required for setting IP addresses on an IOS router.

1. Set IP addresses on the interfaces on R1:

```
R1# configure terminal
R1(config)# interface loopback 0
R1(config-if)# ip address 1.1.1.1 255.255.255.255
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/0
R1(config-if)# ip address 10.42.13.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/1
R1(config-if)# ip address 10.42.12.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# end
R1#
```

2. Set IP addresses on the interfaces on R2:

```
R2# configure terminal
R2(config)# interface loopback 0
R2(config-if)# ip address 2.2.2.2 255.255.255.255
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/0
R2(config-if)# ip address 10.42.12.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/1
R2(config-if)# ip address 10.42.23.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# end
R2#
```

3. Set IP addresses on the interfaces on R3:

```
R3# configure terminal
R3(config)# interface loopback 0
R3(config-if)# ip address 3.3.3.3 255.255.255.255
R3(config-if)# no shutdown
R3(config-if)# interface FastEthernet 0/0
R3(config-if)# ip address 10.42.13.3 255.255.255.0
R3(config-if)# no shutdown
R3(config-if)# interface FastEthernet 0/1
R3(config-if)# ip address 10.42.23.3 255.255.255.0
R3(config-if)# no shutdown
R3(config-if)# end
R3#
```

To Verify the IOS Routers for Initial Connectivity

In order to verify that IP addresses have been configured correctly, a simple ping can test to see if the remote end of a link is reachable and active.

1. Ping R1 to R2:

```
R1#ping 10.42.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

2. Ping R1 to R3:

```
R1#ping 10.42.13.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.13.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

3. Ping R2 to R3:

```
R2#ping 10.42.23.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R2#
```

To Configure the IOS Routers for OSPF

In order to have full reachability in any network, a router must know about every network. This is commonly done by using some sort of Interior Gateway Protocol (IGP), such as Open Shortest Path First (OSPF).

OSPF sends out updates in the form of Link-State Advertisements (LSAs), which all routers in an area use to build their own picture of a network.

MORE? For more information on configuring OSPF, see the *Day One* book: *Advanced OSPF In The Enterprise* at http://www.juniper.net/dayone.

1. Configure the OSPF process and assign interfaces to area 0 on R1:

```
R1# configure terminal
R1(config)# router ospf 1
```

```
R1(config-router)# network 10.42.12.1 0.0.0.0 area 0
R1(config-router)# network 10.42.13.1 0.0.0.0 area 0
R1(config-router# end
R1#
```

2. Configure the OSPF process and assign interfaces to area 0 on R2:

```
R2# configure terminal
R2(config)# router ospf 1
R2(config-router)# network 10.42.12.2 0.0.0.0 area 0
R2(config-router)# network 10.42.23.2 0.0.0.0 area 0
R2(config-router# end
R2#
```

3. Configure the OSPF process and assign interfaces to area 0 on R3:

```
R3# configure terminal
R3(config)# router ospf 1
R3(config-router)# network 10.42.13.3 0.0.0.0 area 0
R3(config-router)# network 10.42.23.3 0.0.0.0 area 0
R3(config-router# end
R3#
```

To Verify OSPF Configuration

Verification that OSPF has been configured correctly can be done by checking to make sure OSPF neighbor adjacencies have formed correctly. Active OSPF neighbors should appear in the FULL state.

1. Verify OSPF neighbor adjacencies on R1:

```
R1#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
2.2.2.2           1   FULL/BDR        00:00:32    10.42.12.2       FastEthernet0/1
3.3.3.3           1   FULL/DR         00:00:34    10.42.13.3       FastEthernet0/0
R1#
```

2. Verify OSPF neighbor adjacencies on R2:

```
R2#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address       Interface
1.1.1.1           1   FULL/DROTHER    00:00:30    10.42.12.1    FastEthernet0/0
3.3.3.3           1   FULL/BDR         00:00:33    10.42.23.3    FastEthernet0/1
R2#
```

3. Verify OSPF neighbor adjacencies on R3:

```
R3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address       Interface
1.1.1.1           1   FULL/DROTHER    00:00:37    10.42.13.1    FastEthernet0/0
2.2.2.2           1   FULL/BDR         00:00:35    10.42.23.2    FastEthernet0/1
R3#
```

To Advertise Loopback Interfaces in OSPF and Ensure They are Passive

Advertising the loopback interfaces into OSPF is an optional step, and is done in this book in order to provide simple verification of a properly functioning OSPF domain. In a production network, the loopback interfaces are often advertised into the OSPF network so that the router-id of the device is a reachable address.

Including the `passive-interface` command simply tells the software to advertise the link as a Type-1 LSA, but it does not attempt to form an adjacency over that interface.

1. Configure OSPF on R1 to include the loopback interface. Include the passive-interface command since there will never be an OSPF neighbor via the loopback:

```
R1# configure terminal
R1(config)# router ospf 1
R1(config-router)# network 1.1.1.1 0.0.0.0 area 0
R1(config-router)# passive-interface lo0
R1(config-router# end
R1#
```

2. Next, configure OSPF on R2's loopback as passive:

```
R2# configure terminal
R2(config)# router ospf 1
R2(config-router)# network 2.2.2.2 0.0.0.0 area 0
R2(config-router)# passive-interface lo0
R2(config-router# end
R2#
```

3. Finally, configure OSPF on R3's loopback interface and set it as passive:

```
R3# configure terminal
R3(config)# router ospf 1
R3(config-router)# network 3.3.3.3 0.0.0.0 area 0
R3(config-router)# passive-interface lo0
R3(config-router# end
R3#
```

To Verify Loopback Interfaces in OSPF

Verification of the loopback interfaces is done simply by checking that the loopback interface is indeed running OSPF, and then by checking the RIB on the other routers to make sure the loopback interfaces are being learned on the other devices.

Alternately, a simple check of the LSDB on any router within the area should give similar information.

1. Ensure the loopback on R1 (1.1.1.1/32) is configured for OSPF, and is being learned on R2 and R3:

```
R1#show ip ospf interface lo0
Loopback0 is up, line protocol is up
  Internet Address 1.1.1.1/32, Area 0
  Process ID 1, Router ID 1.1.1.1, Network Type LOOPBACK, Cost: 1
  Loopback interface is treated as a stub Host
R1#

R2#show ip route 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.12.1 on FastEthernet0/0, 00:02:41 ago
  Routing Descriptor Blocks:
  * 10.42.12.1, from 1.1.1.1, 00:02:41 ago, via FastEthernet0/0
      Route metric is 11, traffic share count is 1
R2#

R3#show ip route 1.1.1.1
Routing entry for 1.1.1.1/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.13.1 on FastEthernet0/0, 00:03:12 ago
  Routing Descriptor Blocks:
  * 10.42.13.1, from 1.1.1.1, 00:03:12 ago, via FastEthernet0/0
      Route metric is 11, traffic share count is 1
R3#
```

2. Ensure the loopback on R2 (2.2.2.2/32) is configured for OSPF, and is being learned on R1 and R3:

```
R2#show ip ospf interface lo0
Loopback0 is up, line protocol is up
  Internet Address 2.2.2.2/32, Area 0
  Process ID 1, Router ID 2.2.2.2, Network Type LOOPBACK, Cost: 1
  Loopback interface is treated as a stub Host
R2#

R1#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.12.2 on FastEthernet0/1, 00:01:58 ago
  Routing Descriptor Blocks:
  * 10.42.12.2, from 2.2.2.2, 00:01:58 ago, via FastEthernet0/1
      Route metric is 11, traffic share count is 1

R1#

R3#show ip route 2.2.2.2
Routing entry for 2.2.2.2/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.23.2 on FastEthernet0/1, 00:02:16 ago
  Routing Descriptor Blocks:
```

```
  * 10.42.23.2, from 2.2.2.2, 00:02:16 ago, via FastEthernet0/1
      Route metric is 11, traffic share count is 1

R3#
```

3. Ensure the loopback on R3 (3.3.3.3/32) is configured for OSPF, and is being learned on R1 and R2:

```
R3#show ip ospf int lo0
Loopback0 is up, line protocol is up
  Internet Address 3.3.3.3/32, Area 0
  Process ID 1, Router ID 3.3.3.3, Network Type LOOPBACK, Cost: 1
  Loopback interface is treated as a stub Host
R3#

R1#show ip route 3.3.3.3
Routing entry for 3.3.3.3/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.13.3 on FastEthernet0/0, 00:04:36 ago
  Routing Descriptor Blocks:
  * 10.42.13.3, from 3.3.3.3, 00:04:36 ago, via FastEthernet0/0
      Route metric is 11, traffic share count is 1

R1#

R2#show ip route 3.3.3.3
Routing entry for 3.3.3.3/32
  Known via "ospf 1", distance 110, metric 11, type intra area
  Last update from 10.42.23.3 on FastEthernet0/1, 00:04:50 ago
  Routing Descriptor Blocks:
  * 10.42.23.3, from 3.3.3.3, 00:04:50 ago, via FastEthernet0/1
      Route metric is 11, traffic share count is 1

R2#
```

To Configure the OSPF Router-id Manually

Sometimes it's beneficial to manually configure a router-id. An example of this is if a router-id that is not an IP address assigned to any interface on the router is desired. It is also sometimes desirable to ensure deterministic entries in the OSPF link-state database (LSDB) as opposed to allowing the system to arbitrarily choose the router-id.

NOTE The OSPF router-id is a 32 bit number, represented in dotted-decimal format (similar to the notation for an IP address). However, like the OSPF area ID, the router ID is not necessarily an IP address, though it is often configured to match an IP address on the device.

1. Use show commands to view the current router-id for R1. Then, configure R1 to use router-id 11.11.11.11:

```
R1#show ip protocols | include ID
  Router ID 1.1.1.1
R1#configure terminal
R1(config)#router ospf 1
R1(config-router)#router-id 11.11.11.11
Reload or use "clear ip ospf process" command, for this to take effect
R1(config-router)#end
R1#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
R1#
```

2. Use show commands to view the current router-id for R2. Then, configure R2 to use router-id 22.22.22.22:

```
R2#show ip protocols | include ID
  Router ID 2.2.2.2
R2#configure terminal
R2(config)#router ospf 1
R2(config-router)#router-id 22.22.22.22
Reload or use "clear ip ospf process" command, for this to take effect
R2(config-router)#end
R2#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
R2#
```

3. Use show commands to view the current router-id for R3. Then, configure R3 to use router-id 33.33.33.33:

```
R3#show ip protocols | include ID
  Router ID 3.3.3.3
R3#configure terminal
R3(config)#router ospf 1
R3(config-router)#router-id 33.33.33.33
Reload or use "clear ip ospf process" command, for this to take effect
R3(config-router)#end
R3#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
R3#
```

To Verify OSPF Router-id Has Been Manually Assigned

You can verify that the router-id has been correctly assigned by checking the output of `show ip protocol`, and can further verify it's correct by checking that the router ID of the adjacencies has been changed.

1. Use show commands to verify the router-id on R1:

```
R1#show ip protocols | include ID
  Router ID 11.11.11.11
R1#
R1#show ip ospf neighbor
```

```
Neighbor ID     Pri   State           Dead Time   Address         Interface
33.33.33.33       1   FULL/BDR        00:00:33    10.42.13.3      FastEthernet0/0
22.22.22.22       1   FULL/BDR        00:00:36    10.42.12.2      FastEthernet0/1
R1#
```

2. Use show commands to verify the router-id on R2:

```
R2#show ip protocols | include ID
  Router ID 22.22.22.22
R2#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
33.33.33.33       1   FULL/BDR        00:00:31    10.42.23.3      FastEthernet0/1
11.11.11.11       1   FULL/DR         00:00:39    10.42.12.1      FastEthernet0/0
R2#
```

3. And, use show commands to verify the router-id on R3:

```
R3#show ip protocols | include ID
  Router ID 33.33.33.33
R3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
22.22.22.22       1   FULL/DR         00:00:30    10.42.23.2      FastEthernet0/1
11.11.11.11       1   FULL/DR         00:00:36    10.42.13.1      FastEthernet0/0
R3#
```

Junos Configuration

Now let's configure the same topology in Junos. Pay attention to the logical structure of the configuration, and the placement of interface and protocol configuration. The advantages of such a simple and standardized structure should be easily apparent.

To Configure the Junos Routers for Initial Connectivity

Here you'll use the knowledge you gained in Chapter 1, and apply interface addressing for each of the routers being configured, including the loopback interface.

1. Set IP addresses on the interfaces on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set interfaces fe-0/0/0.0 family inet address 10.42.12.1/24

[edit]
cjones@R1# set interfaces fe-0/0/1.0 family inet address 10.42.13.1/24

[edit]
cjones@R1# set interfaces lo0.0 family inet address 1.1.1.1/32
```

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Set IP addresses on the interfaces on R2:

```
[edit]
cjones@R2# set interfaces fe-0/0/0.0 family inet address 10.42.23.2/24

[edit]
cjones@R2# set interfaces fe-0/0/1.0 family inet address 10.42.12.2/24

[edit]
cjones@R2# set interfaces lo0.0 family inet address 2.2.2.2/32

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

3. Set IP addresses on the interfaces on R3:

```
[edit]
cjones@R3# set interfaces fe-0/0/0.0 family inet address 10.42.13.3/24

[edit]
cjones@R3# set interfaces fe-0/0/1.0 family inet address 10.42.23.3/24

[edit]
cjones@R3# set interfaces lo0.0 family inet address 3.3.3.3/32

[edit]
cjones@R3# commit and-quit
commit complete
Exiting configuration mode
```

To Verify the Junos Routers for Initial Connectivity

1. Ping from R1 to R2:

```
cjones@R1> ping 10.42.12.2 rapid
PING 10.42.12.2 (10.42.12.2): 56 data bytes
!!!!!
--- 10.42.12.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.238/2.791/4.261/0.752 ms
```

2. Ping from R1 to R3:

```
cjones@R1> ping 10.42.13.3 rapid
PING 10.42.13.3 (10.42.13.3): 56 data bytes
!!!!!
--- 10.42.13.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.332/5.504/17.629/6.063 ms
```

3. And finally, ping from R2 to R3:

```
cjones@R1> ping 10.42.23.3 rapid
PING 10.42.23.3 (10.42.23.3): 56 data bytes
!!!!!
--- 10.42.23.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.042/7.932/30.717/11.394 ms
```

To Configure the Junos Routers for OSPF

Configuring OSPF in Junos is relatively painless, as the only command required is to add each interface to the OSPF hierarchy. You'll notice that in Junos the interface on which to run OSPF is defined under the protocols stanza. This is significantly less complex than the archaic network commands that are used in IOS.

1. Configure the OSPF process and assign interfaces to area 0 on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set protocols ospf area 0 interface fe-0/0/0.0

[edit]
cjones@R1# set protocols ospf area 0 interface fe-0/0/1.0

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Configure the OSPF process and assign interfaces to area 0 on R2:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set protocols ospf area 0 interface fe-0/0/0.0

[edit]
cjones@R2# set protocols ospf area 0 interface fe-0/0/1.0

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

3. Configure the OSPF process and assign interfaces to area 0 on R3:

```
cjones@R3> configure
Entering configuration mode
```

```
[edit]
cjones@R3# set protocols ospf area 0 interface fe-0/0/0.0

[edit]
cjones@R3# set protocols ospf area 0 interface fe-0/0/1.0

[edit]
cjones@R3# commit and-quit
commit complete
Exiting configuration mode
```

To Verify OSPF Configuration

The easiest way to verify that OSPF has been configured correctly is to verify that the expected adjacencies are present. This output should be familiar, as it is very similar to the output of the IOS equivalent command.

1. Verify OSPF neighbor adjacencies on R1:

```
cjones@R1> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.42.12.2       fe-0/0/0.0             Full      2.2.2.2          128    31
10.42.13.3       fe-0/0/1.0             Full      3.3.3.3          128    34
```

2. Verify OSPF neighbor adjacencies on R2:

```
cjones@R2> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.42.23.3       fe-0/0/0.0             Full      3.3.3.3          128    38
10.42.12.1       fe-0/0/1.0             Full      1.1.1.1          128    39
```

3. Verify OSPF neighbor adjacencies on R3:

```
cjones@R3> show ospf neighbor
Address          Interface              State     ID               Pri  Dead
10.42.13.1       fe-0/0/0.0             Full      1.1.1.1          128    39
10.42.23.2       fe-0/0/1.0             Full      2.2.2.2          128    34
```

To Advertise Loopback Interfaces in OSPF and Ensure They are Passive

1. Configure OSPF on R1 to include the loopback interface. Including the passive statement is an optional step, since there will never be an OSPF neighbor via the loopback:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Next, configure OSPF on R2's loopback as passive:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

3. Finally, configure OSPF on R3's loopback interface and set it as passive:

```
cjones@R3> configure
Entering configuration mode

[edit]
cjones@R3# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

To Verify Loopback Interfaces in OSPF

Verification of the loopback's advertisement into OSPF in Junos can be completed both by checking the status of OSPF on the loopback interface, and by checking the routing tables of the other routers in the area.

This is the first time that you've seen the routing table in Junos in this book. As there is no concept of classful networking in Junos, notice that there are no statements similar to the "x.x.x.x/xx is subnetted" that appear in IOS.

1. Ensure the loopback on R1 (1.1.1.1/32) is configured for OSPF, and is being learned on R2 and R3:

```
cjones@R1> show ospf interface brief | match lo0.0
lo0.0              DRother 0.0.0.0         0.0.0.0         0.0.0.0            0

cjones@R2> show route protocol ospf terse 1.1.1.1

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 1.1.1.1/32         O  10          1            >10.42.12.1
```

```
cjones@R3> show route protocol ospf terse 1.1.1.1

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 1.1.1.1/32         O  10          1            >10.42.13.1
```

2. Ensure the loopback on R2 (2.2.2.2/32) is configured for OSPF, and is being learned on R1 and R3:

```
cjones@R2> show ospf interface brief | match lo0.0
lo0.0               DRother 0.0.0.0         0.0.0.0         0.0.0.0            0

cjones@R1> show route protocol ospf terse 2.2.2.2

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 2.2.2.2/32         O  10          1            >10.42.12.2

cjones@R3> show route protocol ospf terse 2.2.2.2

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 2.2.2.2/32         O  10          1            >10.42.23.2
```

3. Ensure the loopback on R3 (3.3.3.3/32) is configured for OSPF, and is being learned on R1 and R2:

```
cjones@R3> show ospf interface brief | match lo0.0
lo0.0               DRother 0.0.0.0         0.0.0.0         0.0.0.0            0

cjones@R1> show route protocol ospf terse 3.3.3.3

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 3.3.3.3/32         O  10          1            >10.42.13.3

cjones@R2> show route protocol ospf terse 3.3.3.3

inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 3.3.3.3/32         O  10          1            >10.42.23.3
```

To Configure the OSPF Router-id Manually

The Junos software will assign a router-id from the first interface that is detected with a non-martian address when rpd starts (typically at boot). This almost always results in the router choosing the address assigned to the loopback interface to be used as the router-id. This is essentially the same behavior that you'd be used to with IOS.

1. Use show commands to view the current router-id for R1. Then, configure R1 to use router-id 11.11.11.11:

```
cjones@R1> show ospf overview | match "Router ID"
  Router ID: 1.1.1.1
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options router-id 11.11.11.11

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Use show commands to view the current router-id for R2. Then, configure R2 to use router-id 22.22.22.22:

```
cjones@R2> show ospf overview | match "Router ID"
  Router ID: 2.2.2.2

cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set routing-options router-id 22.22.22.22

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

3. Use show commands to view the current router-id for R3. Then, configure R3 to use router-id 33.33.33.33:

```
cjones@R3> show ospf overview | match "Router ID"
  Router ID: 3.3.3.3

cjones@R3> configure
Entering configuration mode

[edit]
cjones@R3# set routing-options router-id 33.33.33.33

[edit]
```

```
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

To Verify OSPF Router-id Has Been Manually Assigned

1. Use show commands to verify the router-id on R1:

```
cjones@R1> show ospf overview | match "Router ID"
  Router ID: 11.11.11.11
```

2. Use show commands to verify the router-id on R2:

```
cjones@R2> show ospf overview | match "Router ID"
  Router ID: 22.22.22.22
```

3. And use show commands to verify the router-id on R3:

```
cjones@R3> show ospf overview | match "Router ID"
  Router ID: 33.33.33.33
```

Summary

The task in this section was a simple OSPF network configuration, but it contrasted the almost-archaic network commands for dictating which interfaces on which to run OSPF in IOS with how Junos simplifies the process by allowing an administrator to specify which interfaces will run OSPF under the Junos protocol hierarchy. And again, these small magnifications would magnify in a much larger network as well.

External BGP

Okay, in this example let's configure a simple External BGP peering session between two Autonomous Systems (AS). Comparing the two OSes you should be able to learn not only the Junos methodology but its consistent framework for policy configuration.

Let's just use two routers that are directly connected. The EBGP peering will use the physical interfaces. The following is the diagram of the topology:

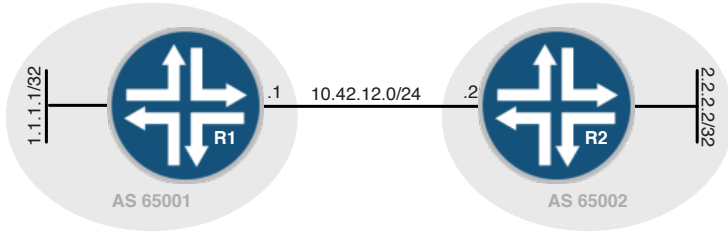

Figure 2.2 External BGP Topology

And our list of tasks in this section is to:

- Configure an EBGP peering between R1 in AS 65001 and R2 in AS 65002.
- Use physical interfaces to peer.
- Advertise loopback interfaces to neighbor.

IOS Configuration

First let's configure this network using IOS.

To Configure the IOS Routers for Initial Connectivity

1. Set IP addresses on the interfaces on R1:

```
R1# configure terminal
R1(config)# interface loopback 0
R1(config-if)# ip address 1.1.1.1 255.255.255.255
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/0
R1(config-if)# ip address 10.42.12.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# end
R1#
```

2. Set IP addresses on the interfaces on R2:

```
R2# configure terminal
R2(config)# interface loopback 0
R2(config-if)# ip address 2.2.2.2 255.255.255.255
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/0
R2(config-if)# ip address 10.42.12.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# end
R2#
```

To Verify Initial Connectivity

1. Ping R1 to R2:

```
R1#ping 10.42.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

To Configure the EBGP Peering Using Physical Interfaces

1. Configure EBGP on R1 to peer with R2 in AS 65002:

```
R1#configure terminal
R1(config)#router bgp 65001
R1(config-router)# no synchronization
R1(config-router)# neighbor 10.42.12.2 remote-as 65002
R1(config-router)# no auto-summary
R1(config-router)#end
R1#
```

2. Configure EBGP on R2 to peer with R1 in AS 65001:

```
R2#configure terminal
R2(config)#router bgp 65002
R2(config-router)# no synchronization
R2(config-router)# neighbor 10.42.12.1 remote-as 65001
R2(config-router)# no auto-summary
R2(config-router)#end
R2#
```

To Verify the EBGP Peering

1. Verify the EBGP peering is in the Established state on R1:

```
R1#show ip bgp neighbors
BGP neighbor is 10.42.12.2,  remote AS 65002, external link
  BGP version 4, remote router ID 2.2.2.2
  BGP state = Established, up for 00:00:30
  Last read 00:00:30, last write 00:00:30, hold time is 180, keepalive interval is 60
seconds
…
```

2. Verify the EBGP peering is in the Established state on R2:

```
R2#show ip bgp neighbors
BGP neighbor is 10.42.12.1,  remote AS 65001, external link
  BGP version 4, remote router ID 1.1.1.1
  BGP state = Established, up for 00:01:57
  Last read 00:00:57, last write 00:00:57, hold time is 180, keepalive interval is 60
seconds
…
```

To Configure EBGP to Advertise the Loopback Addresses

1. Configure the BGP process on R1 to advertise its loopback address:

```
R1#configure terminal
R1(config)#router bgp 65001
R1(config-router)#network 1.1.1.1 mask 255.255.255.255
R1(config-router)#end
R1#
```

2. Configure the BGP process on R2 to advertise its loopback address:

```
R2#configure terminal
R2(config)#router bgp 65002
R2(config-router)#network 2.2.2.2 mask 255.255.255.255
R2(config-router)#end
R2#
```

To Verify EBGP is Advertising the Loopback Addresses

1. Show the BGP neighbor summary on R2 to determine if any BGP prefixes are being received:

```
R2#show ip bgp summary | include Nei|65001
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.42.12.1      4 65001       6       6        3    0    0 00:02:24        1
```

2. Show the received routes on R2:

```
R2#show ip bgp regexp ^65001
BGP table version is 3, local router ID is 2.2.2.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 1.1.1.1/32       10.42.12.1               0             0 65001 i
```

3. Show the RIB on R2:

```
R2#show ip route bgp | e subnetted
B       1.1.1.1 [20/0] via 10.42.12.1, 00:07:17
```

4. Show the BGP neighbor summary on R1:

```
R1#show ip bgp summ | include Nei|65002
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.42.12.2      4 65002       9       9        3    0    0 00:05:16        1
```

5. Show the received routes on R1:

```
R1#show ip bgp regexp ^65002
BGP table version is 3, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 2.2.2.2/32       10.42.12.2               0             0 65002 i
```

6. Show the RIB on R1:

```
R1#show ip route bgp | exclude subnetted
B       2.2.2.2 [20/0] via 10.42.12.2, 00:05:30
```

Junos Configuration

Okay, now let's configure this same network using Junos routers and watch the difference.

To Configure the Junos Routers for Initial Connectivity

1. Set IP addresses on the interfaces on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set interfaces fe-0/0/0.0 family inet address 10.42.12.1/24

[edit]
cjones@R1# set interfaces lo0.0 family inet address 1.1.1.1/32
```

2. Set IP addresses on the interfaces on R2:

```
[edit]
cjones@R1# set interfaces fe-0/0/0.0 family inet address 10.42.12.2/24

[edit]
cjones@R1# set interfaces lo0.0 family inet address 2.2.2.2/32
```

3. And commit the configuration:

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Let's verify the interface configuration on R1:

```
cjones@R1> show configuration interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.42.12.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 1.1.1.1/32;
        }
    }
}
```

5. And verify the interface configuration on R2:

```
cjones@R2> show configuration interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.42.12.2/24;
        }
    }
}
```

```
lo0 {
    unit 0 {
        family inet {
            address 2.2.2.2/32;
        }
    }
}
```

6. And let's verify initial connectivity by pinging R1 to R2:

```
cjones@R1> ping 10.42.12.2 rapid
PING 10.42.12.2 (10.42.12.2): 56 data bytes
!!!!!
--- 10.42.12.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.238/2.791/4.261/0.752 ms
```

To Configure the EBGP Peering Using Physical Interfaces

The biggest change in BGP configuration in Junos, as compared to IOS, is that in Junos the configuration of a BGP neighbor can typically be done in a single command (aside from the AS number).

1. Configure EBGP on R1 to peer with R2 in AS 65002:

```
[edit]
cjones@R1# set routing-options autonomous-system 65001

[edit]
cjones@R1# set protocols bgp group EBGP neighbor 10.42.12.2 peer-as 65002
```

2. Configure EBGP on R2 to peer with R1 in AS 65001:

```
[edit]
cjones@R1# set routing-options autonomous-system 65002

[edit]
cjones@R1# set protocols bgp group EBGP neighbor 10.42.12.1 peer-as 65001
```

3. Commit the configuration:

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Verify the configuration on R1:

```
cjones@R1> show configuration routing-options
autonomous-system 65001;

cjones@R1> show configuration protocols bgp
group EBGP {
    neighbor 10.42.12.2 {
        peer-as 65002;
    }
}
```

5. Verify the configuration on R2:

```
cjones@R2> show configuration routing-options
autonomous-system 65002;

cjones@R2> show configuration protocols bgp
group EBGP {
   neighbor 10.42.12.1 {
      peer-as 65001;
   }
}
```

To Verify the EBGP Peering Using Physical Interfaces

1. First verify the EBGP peering is in the Established state on R1:

```
cjones@R1> show bgp neighbor 10.42.12.2 | match Established
  Type: External    State: Established    Flags: <ImportEval Sync>
```

2. Then verify the EBGP peering is in the Established state on R2:

```
cjones@R2> show bgp neighbor 10.42.12.1 | match Established
  Type: External    State: Established    Flags: <ImportEval Sync>
```

To Configure EBGP to Advertise the Loopback Addresses

Once our BGP session is in the Established state, we must test our configuration. In order to properly verify BGP is working, we must advertise some prefixes to the BGP neighbor. Let's do that now.

Like most routing manipulation in Junos, this is accomplished using policy.

1. Let's create a policy that matches R1's loopback interface, which can then be applied to the BGP process:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R1# set term ADVERTISE_LO0 from protocol direct

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R1# set term ADVERTISE_LO0 from route-filter 1.1.1.1/32 exact

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R1# set term ADVERTISE_LO0 then accept

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R1# top
```

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify policy configuration on R1:

```
cjones@R1> show configuration policy-options
policy-statement EBGP_ADVERTISE_LOOPBACK {
    term ADVERTISE_LO0 {
        from {
            protocol direct;
            route-filter 1.1.1.1/32 exact;
        }
        then accept;
    }
}
```

3. Configure the policy as an *export policy* under the BGP neighbor on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set protocols bgp group EBGP neighbor 10.42.12.2 export EBGP_ADVERTISE_
LOOPBACK

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Verify BGP export policy configuration on R1:

```
cjones@R1> show configuration protocols bgp
group EBGP {
    neighbor 10.42.12.2 {
        export EBGP_ADVERTISE_LOOPBACK;
        peer-as 65002;
    }
}
```

5. Now let's create a policy that matches R2's loopback interface, which can then be applied to the BGP process:

```
[edit]
cjones@R2# edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R2# set term ADVERTISE_LO0 from protocol direct

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R2# set term ADVERTISE_LO0 from route-filter 1.1.1.1/32 exact

[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R2# set term ADVERTISE_LO0 then accept
```

```
[edit policy-options policy-statement EBGP_ADVERTISE_LOOPBACK]
cjones@R2# top

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

6. Verify policy configuration on R2:

```
cjones@R2> show configuration policy-options
policy-statement EBGP_ADVERTISE_LOOPBACK {
    term ADVERTISE_LO0 {
        from {
            protocol direct;
            route-filter 2.2.2.2/32 exact;
        }
        then accept;
    }
}
```

7. Configure the policy as an export policy under the BGP neighbor on R2:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set protocols bgp group EBGP neighbor 10.42.12.1 export EBGP_ADVERTISE_
LOOPBACK

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

8. Verify BGP export policy configuration on R2:

```
cjones@R2> show configuration protocols bgp
group EBGP {
    neighbor 10.42.12.1 {
        export EBGP_ADVERTISE_LOOPBACK;
        peer-as 65001;
    }
}
```

To Verify the Loopback Address is Being Advertised by BGP

Verification of our BGP export policies can be done by examining the BGP summary information, and checking the Adj-RIB-In table (which shows us a list of BGP prefixes received, before import policies, if any, are applied). Also, a quick check of the routing table should confirm that our prefixes are being installed.

1. Let's show the BGP neighbor summary on R1 to determine if any BGP prefixes are being received:

```
cjones@R1> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 1          1          0          0          0          0
Peer         AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.42.12.2   65002   12        13      0      0    4:04 1/1/1/0            0/0/0/0
```

2. Show the received routes on R1:

```
cjones@R1> show route receive-protocol bgp 10.42.12.2

inet.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
  Prefix        Nexthop       MED     Lclpref    AS path
* 2.2.2.2/32   10.42.12.2                         65002 I
```

3. Show the RIB on R1:

```
cjones@R1> show route protocol bgp

inet.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2.2.2.2/32         *[BGP/170] 00:05:22, localpref 100
                      AS path: 65002 I
                    > to 10.42.12.2 via fe-0/0/0.0
```

4. Show the BGP neighbor summary on R2 to determine if any BGP prefixes are being received:

```
cjones@R2> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0                 1          1          0          0          0          0
Peer         AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.42.12.1   65001   15        15      0      0    7:21 1/1/1/0            0/0/0/0
```

5. And last but not least, let's show the received routes on R2:

```
cjones@R2> show route receive-protocol bgp 10.42.12.1

inet.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
  Prefix        Nexthop       MED     Lclpref    AS path
* 1.1.1.1/32   10.42.12.1                         65001 I
```

6. Show the RIB on R2:

```
cjones@R2> show route protocol bgp

inet.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
1.1.1.1/32          *[BGP/170] 00:16:32, localpref 100
                       AS path: 65001 I
                     > to 10.42.12.1 via fe-0/0/0.0
```

Summary

The ability to configure a simple BGP peering with only two commands – setting the autonomous system number and the peering itself – is a clear advantage with Junos. While it is handy to be able to configure route advertisement directly under the BGP process in IOS, there is no question that this is where Junos policy configuration can really shine for you. Having a consistent framework for policy configuration is a clear example of where Junos can excel in your network. As your network grows, or requires new services, these advantages will grow as well.

Another thing worth mentioning is the concept of synchronization, which in IOS states that a route must be actively in the routing table from an IGP before it will be announced via BGP. Typically this is disabled. Junos has no such criteria.

Internal BGP

In this section, let's configure Internal BGP (IBGP). The IBGP peering will be configured to peer between loopback addresses, which will require us to add a static route in order for the remote loopback address to be reachable. IOS Engineers should pay attention to Junos's abilitiy to group BGP peering sessions together.

This section will use two routers directly connected in a single AS, as shown in the diagram of the topology in Figure 2.3.

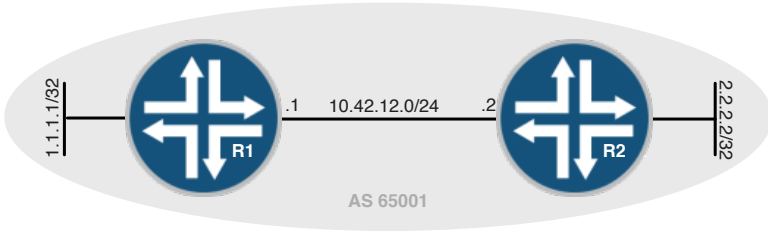

Figure 2.3 Internal BGP Topology

And our list of tasks in this section is to:

- Configure static routes on R1 and R2 for full reachability to the loopback addresses.

- Configure an IBGP peering between R1 and R2 in AS 65001.
- Use loopbacks for IBGP peering.

IOS Configuration

First let's configure this network using IOS, and the sequence should be familiar to you.

To Configure the IOS Routers for Initial Connectivity

1. Set IP addresses on the interfaces on R1:

```
R1# configure terminal
R1(config)# interface loopback 0
R1(config-if)# ip address 1.1.1.1 255.255.255.255
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/0
R1(config-if)# ip address 10.42.12.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# end
R1#
```

2. Set IP addresses on the interfaces on R2:

```
R2# configure terminal
R2(config)# interface loopback 0
R2(config-if)# ip address 2.2.2.2 255.255.255.255
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/0
R2(config-if)# ip address 10.42.12.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# end
R2#
```

To Verify Initial Connectivity

1. Ping R1 to R2:

```
R1#ping 10.42.12.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

To Configure Static Routes for Full Connectivity

1. Configure a static route to R2's loopback address on R1, pointing to R1. R2's physical interface as the next-hop:

```
R1#configure terminal
R1(config)#ip route 2.2.2.2 255.255.255.255 10.42.12.2
```

```
R1(config)#end
R1#
```

2. Configure a static route to R2's loopback address on R1, pointing to R2's physical interface as the next-hop:

```
R2#configure terminal
R2(config)#ip route 1.1.1.1 255.255.255.255 10.42.12.1
R2(config)#end
R2#
```

To Verify the Static Route is Active and Full Connectivity has Been Established

1. Show the RIB on R1:

```
R1#show ip route static | exclude subnett
S      2.2.2.2 [1/0] via 10.42.12.2
R1#
```

2. Show the RIB on R2:

```
R2#show ip route static | exclude subnett
S      1.1.1.1 [1/0] via 10.42.12.1
R2#
```

3. Ping from R1 to R2's loopback:

```
R1#ping 2.2.2.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2.2.2.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/22/24 ms
R1#
```

4. Ping from R2 to R1's loopback:

```
R2#ping 1.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1.1.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 8/16/20 ms
R2#
```

To Configure the IBGP Peering Between Loopbacks

1. Configure IBGP on R1:

```
R1#configure terminal
R1(config)#router bgp 65001
R1(config-router)#no auto-summary
R1(config-router)#no synchronization
R1(config-router)#neighbor 2.2.2.2 remote-as 65001
R1(config-router)#neighbor 2.2.2.2 update-source lo0
R1(config-router)#end
R1#
```

2. Configure IBGP on R2:

```
R2#configure terminal
R2(config)#router bgp 65001
R2(config-router)#no auto-summary
R2(config-router)#no synchronization
R2(config-router)#neighbor 1.1.1.1 remote-as 65001
R2(config-router)#neighbor 1.1.1.1 update-source lo0
R2(config-router)#end
R2#
```

To Verify the IBGP Peering Between Loopbacks

1. Verify the IBGP peering is in the Established state on R1:

```
R1#show ip bgp neighbors
BGP neighbor is 2.2.2.2,  remote AS 65001, internal link
  BGP version 4, remote router ID 2.2.2.2
  BGP state = Established, up for 00:01:01
  Last read 00:00:01, last write 00:00:01, hold time is 180, keepalive interval is 60
seconds
…
```

2. Verify the IBGP peering is in the Established state on R2:

```
R2#show ip bgp neighbors
BGP neighbor is 1.1.1.1,  remote AS 65001, internal link
  BGP version 4, remote router ID 1.1.1.1
  BGP state = Established, up for 00:02:42
  Last read 00:00:42, last write 00:00:42, hold time is 180, keepalive interval is 60
seconds
…
```

Junos Configuration

Now let's configure the same criteria on our Junos routers. You'll notice that the configuration is nearly identical to the EBGP configuration in the previous task. The main difference is the type is now internal. Also, notice how the local-address configuration option is used to specify that the IBGP messages should be sourced from the loopback address.

IOS engineers may notice that an AS number is not configured for an IBGP neighbor in Junos. This is because Junos explicitly defines the BGP peering as internal, whereas in IOS the type is implictly defined as internal by configuring the peer with the same AS as the local router.

To Configure the Junos Routers for Initial Connectivity

1. First, set IP addresses on the interfaces on R1:

```
cjones@R1> configure
Entering configuration mode
```

```
[edit]
cjones@R1# set interfaces fe-0/0/0.0 family inet address 10.42.12.1/24

[edit]
cjones@R1# set interfaces lo0.0 family inet address 1.1.1.1/32
```

2. Then set IP addresses on the interfaces on R2.

```
 [edit]
cjones@R1# set interfaces fe-0/0/0.0 family inet address 10.42.12.2/24

[edit]
cjones@R1# set interfaces lo0.0 family inet address 2.2.2.2/32
```

3. And commit the configuration.

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Let's verify the interface configuration on R1:

```
cjones@R1> show configuration interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.42.12.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 1.1.1.1/32;
        }
    }
}
```

5. And now verify the interface configuration on R2:

```
cjones@R2> show configuration interfaces
fe-0/0/0 {
    unit 0 {
        family inet {
            address 10.42.12.2/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 2.2.2.2/32;
        }
    }
}
```

6. And finally, to verify initial connectivity, let's ping R1 to R2:

```
cjones@R1> ping 10.42.12.2 rapid
PING 10.42.12.2 (10.42.12.2): 56 data bytes
!!!!!
--- 10.42.12.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.238/2.791/4.261/0.752 ms
```

To Configure Static Routes for Full Connectivity

Static routes to the neighbors loopback address are unnecessary in this case, since OSPF is running on the network and there is already full reachability. In this case, we'll configure static routes anyway as doing so demonstrates the syntax required to do so.

1. Configure a static route to R2's loopback address on R1, pointing to R2's physical interface as the next-hop:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options static route 2.2.2.2/32 next-hop 10.42.12.2

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify the static route configuration on R1:

```
cjones@R1> show configuration routing-options
static {
   route 2.2.2.2/32 next-hop 10.42.12.2;
}
```

3. Configure a static route to R1's loopback address on R2, pointing to R1's physical interface as the next-hop:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set routing-options static route 1.1.1.1/32 next-hop 10.42.12.1

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

4. Verify the static route configuration on R2:

```
cjones@R2> show configuration routing-options
```

```
static {
    route 1.1.1.1/32 next-hop 10.42.12.1;
}
```

To Verify the Static Route is Active and Full Connectivity Has Been Established

1. Show the RIB on R1:

```
cjones@R1> show route

inet.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.1.1/32          *[Direct/0] 00:09:23
                     > via lo0.0
2.2.2.2/32          *[Static/5] 00:06:02
                     > to 10.42.12.2 via fe-0/0/0.0
10.42.12.0/24       *[Direct/0] 00:09:23
                     > via fe-0/0/0.0
10.42.12.1/32       *[Local/0] 00:09:23
                        Local via fe-0/0/0.0
```

2. Show the RIB on R2:

```
cjones@R2> show route

inet.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.1.1/32          *[Static/5] 00:07:27
                     > to 10.42.12.1 via fe-0/0/0.0
2.2.2.2/32          *[Direct/0] 00:12:11
                     > via lo0.0
10.42.12.0/24       *[Direct/0] 00:08:16
                     > via fe-0/0/0.0
10.42.12.2/32       *[Local/0] 00:10:04
                        Local via fe-0/0/0.0
```

3. Ping from R1 to R2's loopback:

```
cjones@R1> ping 2.2.2.2 rapid
PING 2.2.2.2 (2.2.2.2): 56 data bytes
!!!!!
--- 2.2.2.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.244/0.370/0.575/0.117 ms
```

4. Ping from R2 to R1's loopback:

```
cjones@R2> ping 1.1.1.1 rapid
PING 1.1.1.1 (1.1.1.1): 56 data bytes
!!!!!
--- 1.1.1.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.374/0.449/0.510/0.045 ms
```

To Configure the IBGP Peering Between Loopbacks

Once again, BGP configuration in Junos is very simple, requiring only the configuration of an AS number and a single command to configure the neighbor.

1. Configure IBGP on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options autonomous-system 65001

[edit]
cjones@R1# set protocols bgp group IBGP type internal neighbor 2.2.2.2 local-address
1.1.1.1

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify IBGP configuration on R1:

```
cjones@R1> show configuration routing-options autonomous-system
65001;

cjones@R1> show configuration protocols bgp
group IBGP {
   type internal;
   neighbor 2.2.2.2 {
      local-address 1.1.1.1;
      peer-as 65001;
   }
}
```

3. Configure IBGP on R2:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set routing-options autonomous-system 65001

[edit]
cjones@R2# set protocols bgp group IBGP type internal neighbor 1.1.1.1 local-address
2.2.2.2

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

To Verify the IBGP Peering Between Loopbacks

Verification of the IBGP is done the same way it is for EBGP, by checking that the peering session is Established.

1. Verify the IBGP peering is in the Established state on R1:

```
cjones@R1> show bgp neighbor 2.2.2.2 | match Established
  Type: Internal    State: Established    Flags: <ImportEval Sync>
```

2. Verify the IBGP peering is in the Established state on R2:

```
cjones@R2> show bgp neighbor 1.1.1.1 | match Established
  Type: Internal    State: Established    Flags: <ImportEval Sync>
```

Summary

Configuring IBGP in Junos is nearly an identical configuration to configuring EBGP. The main differences are that type internal is used, and the peer-as matches the local AS, however the peer-as command is not required and the peer AS and local AS are implied to be the same whenever the type is internal.

One of the advantages of Junos is being able to group BGP peering sessions together. This can provide logical separation of IBGP and EBGP peering sessions, separation of service provider connections, or whatever purpose you can come up with.

Also notice that Junos does not require an equivalent to the `no auto summary` command in IOS. This is because Junos doesn't have the concept of classful or classless networking. CIDR is used exclusively.

VLAN Configuration

Let's perform basic switching configuration tasks to see how the two OSes compare here. Of importance to the IOS Engineer is how in Junos VLANs are named, and that the VLAN-ID tag is then configured under that VLAN.

The topology for this section is a single layer 3 switch, so a diagram is unnecessary.

And our task list in this section is to:

- Configure a VLAN
- Assign the new VLAN to an access (untagged) interface
- Create a trunk (802.1Q tagged) interface
- Create a layer 3 interface

IOS Configuration

First let's configure this network using IOS, a task most all of you should be familiar with.

To Configure a VLAN On an IOS Switch

1. Simply create the VLAN and assign it a name:

```
SW1#configure terminal
SW1(config)#vlan 10
SW1(config-vlan)#name IOS_VLAN
SW1(config-vlan)#exit
SW1(config)#end
SW1#
```

To Verify the VLAN Has Been Created and Named

1. Show the VLAN list:

```
SW1#show vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Gi0/1, Gi0/2
10   IOS_VLAN                         active

SW1#
```

To Assign a VLAN to an Access Interface

1. Configure the interface as an access interface, and assign VLAN 10 to be untagged on it:

```
SW1#configure terminal
SW1(config)#interface fa0/2
SW1(config-if)#switchport mode access
SW1(config-if)#switchport access vlan 10
SW1(config-if)#end
SW1#
```

To Verify the VLAN Has Been Added to fa0/2 as Untagged

1. Show the VLAN list once again:

```
SW1#show vlan br

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Gi0/1, Gi0/2
10   IOS_VLAN                         active    Fa0/2

SW1#
```

To Create a Trunk Interface Carrying All VLANs

1. In IOS, the default is to carry all configured VLANs on a trunk interface. So let's configure an interface as an 802.1Q tagged interface:

```
SW1#configure terminal
SW1(config)#interface fa0/24
SW1(config-if)#switchport mode trunk
SW1(config-if)#switchport trunk encapsulation dot1q
SW1(config-if)#end
SW1#
```

To Verify the Trunk Interface

1. Show the trunk configuration and verify:

```
SW1#show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Fa0/24      on               802.1q         trunking      1

Port        Vlans allowed on trunk
Fa0/24      1-4094

Port        Vlans allowed and active in management domain
Fa0/24      1,10

Port        Vlans in spanning tree forwarding state and not pruned
Fa0/24      1,10
```

To Create a Layer 3 Interface (Known as an SVI in IOS)

1. Configure a L3 interface for IOS_VLAN:

```
SW1#configure terminal
SW1(config)#interface vlan 10
SW1(config-if)#ip address 10.10.10.1 255.255.255.0
SW1(config-if)#end
SW1#
```

To Verify the Layer 3 Interface

1. And to finish the verification process, show the list of interface IP addresses on the switch:

```
SW1#show ip int br | include Vlan
Vlan1                  unassigned      YES NVRAM  administratively down down
Vlan10                 10.10.10.1      YES manual up                    up
```

Junos Configuration

Okay let's move on to the parallel task in Junos. IOS Engineers should notice how the VLAN are named and then configured.

To Configure a VLAN On a Junos Switch

1. Create the VLAN and assign it a VLAN ID:

```
cjones@SW1> configure
Entering configuration mode

[edit]
cjones@SW1# set vlans JUNOS_VLAN vlan-id 10

[edit]
cjones@SW1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify the VLAN configuration:

```
cjones@SW1> show configuration vlans
JUNOS_VLAN {
   vlan-id 10;
}
```

To Verify the VLAN Has Been Created and Named

1. Verify by showing the VLAN list:

```
cjones@SW1> show vlans JUNOS_VLAN
Name         Tag    Interfaces
JUNOS_VLAN    10
                 None
```

To Assign a VLAN to an Access Interface

In Junos there are actually two ways of assigning a VLAN to an interface:

- Under the VLAN configuration
- Under the interface

Let's configure the access VLAN on an interface under the VLAN configuration stanza. Note that in either case, the ethernet-switching family must be configured on the interface.

1. Configure the interface with the ethernet-switching family, and assign VLAN 10 to the VLAN configuration stanza:

```
cjones@SW1> configure
Entering configuration mode

[edit]
cjones@SW1# set interfaces fe-0/0/0 unit 0 family ethernet-switching port-mode access

[edit]
cjones@SW1# set vlans JUNOS_VLAN interface fe-0/0/0.0

[edit]
cjones@SW1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify the access interface configuration:

```
cjones@SW1> show configuration interfaces fe-0/0/0.0
family ethernet-switching;
port-mode access;

cjones@SW1> show configuration vlans
JUNOS_VLAN {
   vlan-id 10;
   interface {
      fe-0/0/0.0;
   }
}
```

To Verify the VLAN Has Been Added to fe-0/0/0.0 as Untagged

1. Show the VLAN list:

```
cjones@SW1> show vlans JUNOS_VLAN
Name         Tag    Interfaces
JUNOS_VLAN    10     fe-0/0/0.0
```

2. Show the extensive output of the VLAN list to verify the VLAN is untagged on fe-0/0/0.0:

```
cjones@SW1> show vlans extensive JUNOS_VLAN
VLAN: JUNOS_VLAN, Created at: Sun Jun 24 12:54:01 2012
802.1Q Tag: 10, Internal index: 2, Admin State: Enabled, Origin: Static
Protocol: Port Mode, Mac aging time: 300 seconds
Number of interfaces: Tagged 0 (Active = 0), Untagged  1 (Active = 0)
     fe-0/0/0.0, untagged, access
```

3. Show the ethernet-switching interfaces detail:

```
cjones@SW1> show ethernet-switching interfaces detail
Interface: fe-0/0/0.0 Index: 69
  State: UP
  Vlans: JUNOS_VLAN(untagged)
```

To Create a Trunk Interface Carrying All VLANs

This time let's configure the VLAN under the interface, rather than the interface under the VLAN. This method will seem more familiar to IOS engineers. It doesn't matter which method you choose, but the important thing is to be consistent in your approach.

1. Configure an interface with the Ethernet-switching family, and tag VLAN JUNOS_VLAN on that interface:

```
cjones@SW1> configure
Entering configuration mode

[edit]
cjones@SW1# set interfaces fe-0/0/0 unit 0 family ethernet-switching port-mode trunk
vlan members all

[edit]
cjones@SW1# commit and-quit
commit complete
Exiting configuration mode
```

2. Verify the trunk configuration:

```
cjones@SW1> show configuration interfaces fe-0/0/0.0
family ethernet-switching {
   port-mode trunk;
   vlan {
      members all;
   }
}
```

To Verify the Trunk Interface

1. Show the VLAN list:

```
cjones@SW1> show vlans JUNOS_VLAN
Name         Tag    Interfaces
JUNOS_VLAN    10
                    fe-0/0/0.0
```

2. Show the extensive output of the VLAN list to verify the VLAN is tagged on fe-0/0/0.0:

```
cjones@SW1> show vlans extensive JUNOS_VLAN
VLAN: JUNOS_VLAN, Created at: Sun Jun 24 12:54:01 2012
802.1Q Tag: 10, Internal index: 2, Admin State: Enabled, Origin: Static
```

```
Protocol: Port Mode, Mac aging time: 300 seconds
Number of interfaces: Tagged 1 (Active = 0), Untagged  0 (Active = 0)
     fe-0/0/0.0, tagged, trunk
```

3. Show the ethernet-switching interfaces detail:

```
cjones@SW1> show ethernet-switching interfaces detail
Interface: fe-0/0/6.0, Index: 84, State: up, Port mode: Trunk
Ether type for the interface: 0x8100
VLAN membership:
   JUNOS_VLAN, 802.1Q Tag: 10, tagged, msti-id: 0, blocked by STP
```

To Create a Layer 3 Interface (Known as an RVI in Junos)

You can see how configuration of a Routed Virtual Interface (RVI) may be a bit strange to an IOS engineer, but in reality it provides a very consistent approach with regard to Junos interface configuration. Junos uses a single VLAN interface, with a unit number. The unit number is arbitrary, but should match the VLAN ID for consistency.

Compare this approach to that of IOS, where a separate interface is created for each VLAN.

1. Configure the L3 interface under the VLAN:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set vlans JUNOS_VLAN l3-interface vlan.10
```

2. Configure the RVI (VLAN interface). Convention states that the unit number under the VLAN interface should match the VLAN ID. While this is not a requirement, it is certainly a best practice.

```
[edit]
cjones@R1# set interfaces vlan unit 10 family inet address 10.10.10.1/24
```

3. Commit the configuration:

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Verify the configuration:

```
cjones@R1> show configuration vlans
JUNOS_VLAN {
   vlan-id 10;
   l3-interface vlan.10;
}

cjones@R1> show configuration interfaces vlan
```

```
unit 10 {
   family inet {
      address 10.10.10.1/24;
   }
}
```

To Verify the Layer 3 Interface

1. Show the list of interface IP addresses on the switch and you can see the verification:

```
cjones@R1> show interfaces terse | match vlan
vlan                    up    up
vlan.10                 up    up   inet     10.10.10.1/24
```

Summary

The simplicity of configuring VLANs, and Layer-3 interfaces for those VLANs, is fairly equal between Junos and IOS. The key point here for IOS Engineers is that while IOS configuration creates the VLAN ID and allows you to name that VLAN, Junos VLANs are named and the VLAN-ID tag is configured under that VLAN.

Simple NAT

In this section, let's configure the two most common forms of NAT that one might encounter in an enterprise: source NAT to an outside interface address, and destination NAT to an inside host. The two OSes are equally paired in this match, so it's simply a matter of the process the IOS Engineer should pay attention to. This is true for most of the book, which should put to bed the rumor that Junos is somehow more complicated than IOS. It isn't.

This section will use two routers directly connected in a single AS, as shown in the topology of Figure 2.4.

And our task list in this section is to:

- Source NAT 10.42.0.0/24 to use the outside interface (200.200.200.1).
- Destination NAT on 200.200.200.1 port 80 to the web server at 10.42.0.100 running on port 80.

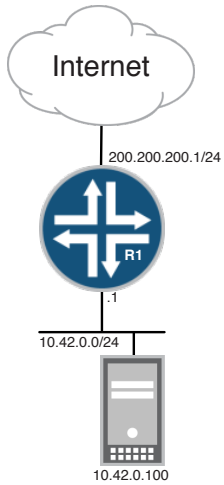

Figure 2.4 NAT Topology

IOS Configuration

To Configure Source NAT for 10.42.0.0/24 in IOS

```
R1#configure terminal
R1(config)#interface f0/0
R1(config-if)#ip address 200.200.200.1 255.255.255.0
R1(config-if)#ip nat outside
R1(config-if)#interface f0/1
R1(config-if)#ip address 10.42.0.1 255.255.255.0
R1(config-if)#ip nat inside
R1(config-if)#exit
R1(config)#access-list 1 permit 10.42.0.0 0.0.0.255
R1(config)#ip nat inside source list 1 interface FastEthernet 0/0 overload
R1(config)#end
R1#
```

To Verify Source NAT for 10.42.0.0/24

```
R1# show ip nat translations
Pro Inside global         Inside local       Outside local      Outside global
tcp 200.200.200.1:53638   10.42.0.2:53638    74.125.225.136:80  74.125.225.136:80
```

To Configure Destination NAT for a Web Server in IOS

```
R1# configure terminal
R1(config)# ip nat inside source static tcp 10.42.0.100 80 200.200.200.1 80 extendable
R1(config)# end
R1#
```

To Verify Destination NAT For a Web Server

```
R1# show ip nat translations
Pro Inside global         Inside local       Outside local      Outside global
tcp 200.200.200.1:80      10.42.0.100:80     ---                ---
```

Junos Configuration

Now let's configure Junos with the same criteria, but in order to configure NAT as shown in this book, the device must be in flow-mode, which is the default on branch SRX devices. While in flow-mode, firewall security policies must be configured in order for traffic to flow. Configuring security policies is beyond the scope of this book.

MORE? For more information on configuring security policies, see *Junos Security* by Rob Cameron, Brad Woodward, et. al., at http://www.juniper.net/books.

To Configure Source NAT for 10.42.0.0/24 in Junos

Source NAT on Junos is very straightforward, using standard policy match/then syntax as well as indicating the security zones involved in the NAT process.

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set interfaces fe-0/0/0 unit 0 family inet address 200.200.200.1/24

[edit]
cjones@R1# set interfaces fe-0/0/1 unit 0 family inet address 10.42.0.1/24

[edit]
cjones@R1# edit security nat source rule-set LAN_RULE

[edit security nat source rule-set LAN_RULE]
cjones@R1# set from zone LAN

[edit security nat source rule-set LAN_RULE]
cjones@R1# set to zone INET

[edit security nat source rule-set LAN_RULE]
cjones@R1# set rule NAT_10.42_24 match source-address 10.42.0.0/24

[edit security nat source rule-set LAN_RULE]
cjones@R1# set rule NAT_10.42_24 match destination-address 0.0.0.0/0

[edit security nat source rule-set LAN_RULE]
cjones@R1# set rule NAT_10.42_24 then source-nat interface
```

```
[edit security nat source rule-set LAN_RULE]
cjones@R1# top

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

The above sequence of set commands creates the following configuration:

```
interfaces {
    fe-0/0/0 {
        unit 0 {
            family inet {
                address 200.200.200.1/24;
            }
        }
    }
    fe-0/0/1 {
        unit 0 {
            family inet {
                address 10.42.0.1/24;
            }
        }
    }
}
security {
    nat {
        source {
            rule-set LAN_RULE {
                from zone LAN;
                to zone INET;
                rule NAT_ALL {
                    match {
                        source-address 10.42.0.0/24;
                        destination-address 0.0.0.0/0;
                    }
                    then {
                        source-nat {
                            interface;
                        }
                    }
                }
            }
        }
    }
}
```

To Verify Source NAT for 10.42.0.0/24

1. Show the source NAT summary:

```
cjones@R1> show security nat source summary
Total port number usage for port translation pool: 0
Maximum port number for port translation pool: 16777216
Total pools: 0

Total rules: 1
Rule name        Rule set      From           To             Action
NAT_ALL          LAN_RULE      LAN            INET           interface
```

2. Show the security flow session for 10.42.0.100:

```
cjones@R1> show security flow session nat source-prefix 10.42.0.100
Session ID: 14582, Policy name: PERMIT_ALL/4, Timeout: 1754, Valid
  In: 10.42.0.100/49401 -> 74.125.239.8/443;tcp, If: fe-0/0/1, Pkts: 10, Bytes: 394
  Out: 74.125.239.8/443 -> 200.200.200.1/15654;tcp, If: fe-0/0/0.0, Pkts: 8, Bytes:
3970

Session ID: 23401, Policy name: PERMIT_ALL/4, Timeout: 1786, Valid
  In: 10.42.0.100/49163 -> 207.17.137.239/80;tcp, If: fe-0/0/1, Pkts: 18, Bytes: 276
  Out: 207.17.137.239/80 -> 200.200.200.1/5973;tcp, If: fe-0/0/0.0, Pkts: 14, Bytes:
353
Total sessions: 2
```

To Configure Destination NAT For a Web Server in Junos

Destination NAT in Junos is very simple and straightfoward. It's easy to visualize exactly what's happening in the configuration, since Junos security policy follows a standard if/then format.

The creation of a destination pool is essential for destination NAT in Junos. The pool specifies the internal IP address that the external request will be sent to, once translated.

A destination NAT policy simply specifies match criteria, and then an action to perform on any matches. In this case, that action is destination-nat pool <poolname>.

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set security nat destination pool WEBSERVER address 10.42.0.100/32

[edit]
cjones@R1# set security nat destination pool WEBSERVER address port 80

[edit]
cjones@R1# edit security nat destination rule-set DESTINATION_NAT
```

```
[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# set from zone INET

[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# set rule DST_NAT_WEBSERVER match source-address 0.0.0.0/0

[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# set rule DST_NAT_WEBSERVER match destination-address 200.200.200.1/32

[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# set rule DST_NAT_WEBSERVER match destination-port 80

[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# set rule DST_NAT_WEBSERVER then destination-nat pool WEBSERVER

[edit security nat destination rule-set DESTINATION_NAT]
cjones@R1# top

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

The above sequence of set commands results in the following configuration:

```
security {
    nat {
        destination {
            pool WEBSERVER {
                address 10.42.0.100/32 port 80;
            }
            rule-set DESTINATION_NAT {
                from zone INET;
                rule DST_NAT_WEBSERVER {
                    match {
                        source-address 0.0.0.0/0;
                        destination-address 200.200.200.1/32;
                        destination-port 80;
                    }
                    then {
                        destination-nat pool WEBSERVER;
                    }
                }
            }
        }
    }
}
```

To Verify Destination NAT for a Web Server

1. Show the destination NAT summary:

```
cjones@R1> show security nat destination summary
Total pools: 1
Pool name       Address                         Routing         Port  Total
                Range                           Instance              Address
WEBSERVER       10.42.0.100 - 10.42.0.100       default         80    1

Total rules: 1
Rule name            Rule set          From                        Action
DST_NAT_WEBSERVER    DESTINATION_NAT   INET                        WEBSERVER
```

2. Show the security session for the destination NAT flow to 10.42.0.100:

```
cjones@R1> show security flow session nat destination-prefix 200.200.200.1
Session ID: 10057, Policy name: PERMIT_HTTP_TO_WEBSERVER/4, Timeout: 1772, Valid
  In: 207.17.137.239/22072 --> 200.200.200.1/80;tcp, If: fe-
0/0/0.0, Pkts: 904, Bytes: 265591
  Out: 10.42.0.100/80 --> 207.17.137.239/22072;tcp, If: fe-
0/0/0.1, Pkts: 879, Bytes: 38638
```

Conclusion

The Junos configuration of NAT is clearly more straightforward, while IOS uses somewhat cryptic names for the IP addresses pre- and post-translation. However, the NAT verification is more easily determined in IOS thanks to the simple-to-understand NAT translation table.

Chapter 3

Case Study

In this chapter, you will learn how many of the configuration examples in this book fit together. A small-scale version of a typical enterprise network will be built and configured using the building blocks from earlier chapters.

Let's use a simple three-router topology, configured as a single-area OSPF network, with EBGP to two service providers who will each be sending a default route. The diagram of the topology can be seen in Figure 3.1.

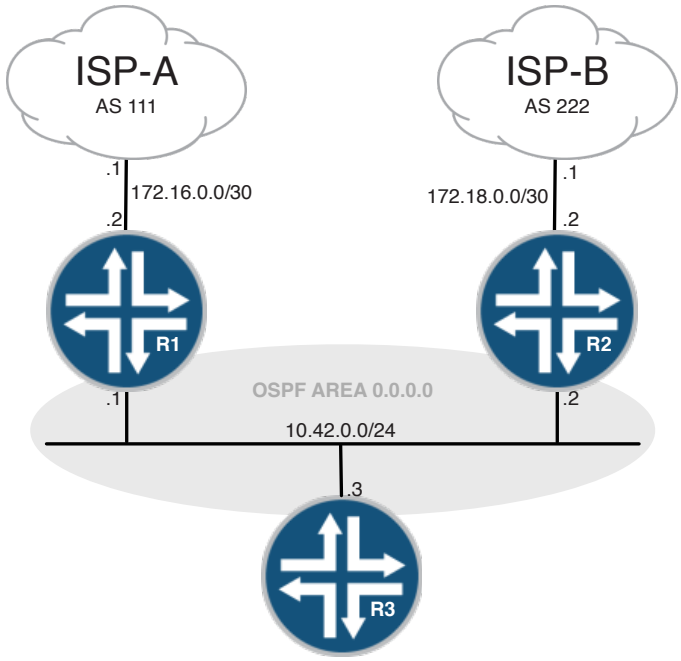

Figure 3.1 Case Study Topology

And our list of tasks in this section is to:

- Configure OSPF between R1, R2, and R3
- Advertise a default route into OSPF on R1 and R2
- Configure EBGP between R1 and ISP-A
- Configure EBGP between R2 and ISP-B
- Configure IBGP between R1 and R2, using loopback interfaces, including a next-hop-self policy
- Prefer inbound and outbound traffic via ISP-B
- Advertise an aggregate prefix of 10.42.0.0/16 towards ISP-A and ISP-B

IOS Configuration

Okay, let's configure this network using IOS. As always, key events are bolded.

To Configure the IOS Routers for Initial Connectivity

1. Set IP addresses on the interfaces on R1:

```
R1# configure terminal
R1(config)# interface loopback 0
R1(config-if)# ip address 1.1.1.1 255.255.255.255
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/0
R1(config-if)# ip address 172.16.0.2 255.255.255.252
R1(config-if)# no shutdown
R1(config-if)# interface FastEthernet 0/1
R1(config-if)# ip address 10.42.0.1 255.255.255.0
R1(config-if)# no shutdown
R1(config-if)# end
R1#
```

2. Set IP addresses on the interfaces on R2:

```
R2# configure terminal
R2(config)# interface loopback 0
R2(config-if)# ip address 2.2.2.2 255.255.255.255
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/0
R2(config-if)# ip address 172.18.0.2 255.255.255.252
R2(config-if)# no shutdown
R2(config-if)# interface FastEthernet 0/1
R2(config-if)# ip address 10.42.0.2 255.255.255.0
R2(config-if)# no shutdown
R2(config-if)# end
R2#
```

3. Set IP addresses on the interfaces on R3:

```
R3# configure terminal
R3(config)# interface loopback 0
R3(config-if)# ip address 3.3.3.3 255.255.255.255
R3(config-if)# no shutdown
R3(config-if)# interface FastEthernet 0/0
R3(config-if)# ip address 10.42.0.3 255.255.255.0
R3(config-if)# no shutdown
R3(config-if)# end
R3#
```

To Verify Initial Connectivity

1. Ping R1 to R2:

```
R1#ping 10.42.0.2
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

2. Ping R1 to R3:

```
R1#ping 10.42.0.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

3. Ping R2 to R3:

```
R2#ping 10.42.0.3

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.42.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R2#
```

4. Ping R1 to ISP-A:

```
R1#ping 172.16.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R1#
```

5. Ping R2 to ISP-B:

```
R2#ping 172.18.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.18.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
R2#
```

To Configure the IOS Routers for OSPF

1. Configure the OSPF process and assign interfaces to area 0 on R3. Configure the loopback interface as passive. Also, manually set the OSPF router-id:

```
R1# configure terminal
R1(config)# router ospf 1
R1(config-router)# router-id 1.1.1.1
R1(config-router)# network 1.1.1.1 0.0.0.0 area 0
R1(config-router)# network 10.42.0.1 0.0.0.0 area 0
R1(config-router)# passive-interface lo0
R1(config-router# end
R1#
```

2. Configure the OSPF process and assign interfaces to area 0 on R3. Also, manually set the OSPF router-id:

```
R2# configure terminal
R2(config)# router ospf 1
R2(config-router)# router-id 2.2.2.2
R2(config-router)# network 2.2.2.2 0.0.0.0 area 0
R2(config-router)# network 10.42.0.2 0.0.0.0 area 0
R2(config-router)# passive-interface lo0
R2(config-router# end
R2#
```

3. Configure the OSPF process and assign interfaces to area 0 on R3. Also, manually set the OSPF router-id:

```
R3# configure terminal
R3(config)# router ospf 1
R3(config-router)# router-id 3.3.3.3
R3(config-router)# network 3.3.3.3 0.0.0.0 area 0
R3(config-router)# network 10.42.0.3 0.0.0.0 area 0
R3(config-router)# passive-interface lo0
R3(config-router# end
R3#
```

To Verify OSPF Configuration

1. Verify OSPF neighbor adjacencies on R1:

```
R1#show ip ospf neigh

Neighbor ID     Pri   State           Dead Time   Address         Interface
2.2.2.2           1   FULL/BDR        00:00:32    10.42.0.2       FastEthernet0/1
3.3.3.3           1   FULL/DR         00:00:34    10.42.0.3       FastEthernet0/1
R1#
```

2. Verify OSPF neighbor adjacencies on R2:

```
R2#show ip ospf neigh

Neighbor ID     Pri   State           Dead Time   Address         Interface
1.1.1.1           1   FULL/DROTHER    00:00:30    10.42.0.1       FastEthernet0/1
3.3.3.3           1   FULL/BDR        00:00:33    10.42.0.3       FastEthernet0/1
R2#
```

3. Verify OSPF neighbor adjacencies on R3:

```
R3#show ip ospf neigh

Neighbor ID     Pri   State           Dead Time   Address         Interface
1.1.1.1           1   FULL/DROTHER    00:00:37    10.42.0.1       FastEthernet0/0
2.2.2.2           1   FULL/BDR        00:00:35    10.42.0.2       FastEthernet0/0
R3#
```

To Configure R1 and R2 to Inject a Default Route Into OSPF in IOS

1. Configure R1 to inject a default route into the OSPF process:

```
R1#configure terminal
R1(config)#router ospf 1
R1(config-router)#default-information originate
R1(config-router)#end
R1#
```

2. Configure R2 to inject a default route into the OSPF process:

```
R2#configure terminal
R2(config)#router ospf 1
R2(config-router)#default-information originate
R2(config-router)#end
R2#
```

To Verify the OSPF Default Route in IOS

1. Check for the OSPF type-5 LSA in the OSPF database on R3:

```
R3#show ip ospf database external

          OSPF Router with ID (3.3.3.3) (Process ID 1)

            Type-5 AS External Link States

  Routing Bit Set on this LSA
  LS age: 164
  Options: (No TOS-capability, DC)
  LS Type: AS External Link
  Link State ID: 0.0.0.0 (External Network Number )
  Advertising Router: 1.1.1.1
  LS Seq Number: 80000001
  Checksum: 0x1D91
  Length: 36
  Network Mask: /0
          Metric Type: 2 (Larger than any link state path)
          TOS: 0
          Metric: 1
          Forward Address: 0.0.0.0
          External Route Tag: 1
```

```
  Routing Bit Set on this LSA
  LS age: 82
  Options: (No TOS-capability, DC)
  LS Type: AS External Link
  Link State ID: 0.0.0.0 (External Network Number )
  Advertising Router: 2.2.2.2
  LS Seq Number: 80000001
  Checksum: 0xFEAB
  Length: 36
  Network Mask: /0
          Metric Type: 2 (Larger than any link state path)
          TOS: 0
          Metric: 1
          Forward Address: 0.0.0.0
          External Route Tag: 1
```

2. Check for the OSPF external route in the RIB:

```
R3#show ip route ospf
    1.0.0.0/32 is subnetted, 1 subnets
O      1.1.1.1 [110/11] via 10.42.0.1, 00:59:55, FastEthernet0/0
    2.0.0.0/32 is subnetted, 1 subnets
O      2.2.2.2 [110/11] via 10.42.0.2, 00:59:55, FastEthernet0/0
O*E2 0.0.0.0/0 [110/1] via 10.42.0.2, 00:03:00, FastEthernet0/0
             [110/1] via 10.42.0.1, 00:04:22, FastEthernet0/0
```

To Configure IOS Routers R1 and R2 for EBGP to ISP-A and ISP-B

1. Configure R1 (AS 65001) to peer with ISP-A (AS 111):

```
R1# configure terminal
R1(config)# router bgp 65001
R1(config-router)# no synchronization
R1(config-router)# neighbor 172.16.0.1 remote-as 111
R1(config-router)# no auto-summary
R1(config-router)# end
R1#
```

2. Configure R2 (AS 65001) to peer with ISP-B (AS 222):

```
R2# configure terminal
R2(config)# router bgp 65001
R2(config-router)# no synchronization
R2(config-router)# neighbor 172.18.0.1 remote-as 222
R2(config-router)# no auto-summary
R2(config-router)# end
R2#
```

To Verify EBGP Configuration on IOS

1. Verify R1's EBGP peering with ISP-A:

```
R1#show ip bgp neighbors 172.16.0.1 | include =
  BGP state = Established, up for 00:08:37
```

```
R1#show ip bgp summary | include 172.16.0.1|Neighbor
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
172.16.0.1      4   111      11      10        2    0    0 00:07:50        1
R1#
```

2. Verify R1 is receiving the default route advertised by ISP-A by inspecting the Adj-RIB-In table (also known as the BGP table) on R1:

```
R1#show ip bgp
BGP table version is 2, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          172.16.0.1               0             0 111 i
```

3. Verify the BGP-learned default route is being installed in the RIB (also known as the routing table):

```
R1#show ip route bgp
B*   0.0.0.0/0 [20/0] via 172.16.0.1, 00:23:52
```

4. Verify R2's EBGP peering with ISP-B:

```
R2#show ip bgp neighbors 172.18.0.1 | include =
  BGP state = Established, up for 00:02:46

R2#show ip bgp summary | include 172.18.0.1|Neighbor
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
172.18.0.1      4   222       7       6        2    0    0 00:03:37        1
```

5. Verify R2 is receiving the default route advertised by ISP-B by inspecting the Adj-RIB-In table on R2:

```
R2#show ip bgp
BGP table version is 2, local router ID is 2.2.2.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          172.18.0.1               0             0 222 i
```

6. Verify the BGP-learned default route is being installed in the RIB (also known as the routing table):

```
R2#show ip route bgp
B*   0.0.0.0/0 [20/0] via 172.18.0.1, 00:21:23
```

To Configure IOS Routers R1 and R2 for IBGP

1. Configure R1 for IBGP to R2, using the loopback interface for peering:

```
R1# configure terminal
R1(config)# router bgp 65001
R1(config-router)# neighbor 2.2.2.2 remote-as 65001
R1(config-router)# neighbor 2.2.2.2 update-source Loopback0
R1(config-router)# end
R1#
```

2. Configure a static route on R1 to enable full reachability to R2's loopback. Since OSPF is running, this step isn't necessary, but let's configure it anyway for demonstration purposes. Since it is preferred to use the OSPF route, give the static route a high route preference (known as *administrative distance* in IOS):

```
R1# configure terminal
R1(config)#ip route 2.2.2.2 255.255.255.255 10.42.0.2 254
R1(config)#end
R1#
```

3. Configure the IBGP peering on R1 to update the next-hop of all prefixes sent to R2:

```
R1#configure terminal
R1(config)#router bgp 65001
R1(config-router)#neighbor 2.2.2.2 next-hop-self
R1(config-router)#end
R1#
```

4. Configure R2 for IBGP to R1, using the loopback interface for peering:

```
R2#configure terminal
R2(config)#router bgp 65001
R2(config-router)#neighbor 1.1.1.1 remote-as 65001
R2(config-router)#neighbor 1.1.1.1 update-source Loopback0
R2(config)#end
R2#
```

5. Configure a static route on R2 to enable full reachability to R1's 5. loopback in the event of an OSPF failure:

```
R2#configure terminal
R2(config)#ip route 1.1.1.1 255.255.255.255 10.42.0.1 254
R2(config)#end
R2#
```

6. Configure the IBGP peering on R2 to update the next-hop of all prefixes sent to R1:

```
R2#configure terminal
R2(config)#router bgp 65001
R2(config-router)#neighbor 1.1.1.1 next-hop-self
R2(config-router)#end
R2#
```

To Verify IBGP Configuration Between R1 and R2

1. Verify the IBGP adjacency between R1 and R2:

```
R1#show ip bgp neighbors 2.2.2.2 | include =
  BGP state = Established, up for 00:09:14
```

2. Verify R1 is receiving BGP prefixes from R2:

```
R1#show ip bgp summary | include 2.2.2.2|Neighbor
Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
2.2.2.2         4 65001      12      11        2    0    0 00:06:30        1
```

To Advertise the Aggregate Prefix 10.42.0.0/16 to AS 111 and AS 222

1. Add the aggregate-address command under the BGP process on R1:

```
R1#configure terminal
R1(config)#router bgp 65001
R1(config-router)# aggregate-address 10.42.0.0 255.255.0.0 summary-only
R1(config-router)# network 10.42.0.0 mask 255.255.255.0
R1(config-router)# end
R1#
```

2. Add the aggregate-address command under the BGP process on R2:

```
R2#configure terminal
R2(config)#router bgp 65001
R2(config-router)# aggregate-address 10.42.0.0 255.255.0.0 summary-only
R2(config-router)# network 10.42.0.0 mask 255.255.255.0
R2(config-router)# end
R2#
```

To Verify the Aggregate is Being Sent to AS 111 and AS 222

1. Check the list of routes being advertised from R1 to ISP-A:

```
R1#show ip bgp neighbors 172.16.0.1 advertised-routes

BGP table version is 5, local router ID is 1.1.1.1

Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale

Origin codes: i - IGP, e - EGP, ? - incomplete
```

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.42.0.0/16     0.0.0.0                              32768 i

Total number of prefixes 1
```

2. Check the list of routes being advertised from R2 to ISP-B:

```
R2#show ip bgp neighbors 172.18.0.1 advertised-routes
BGP table version is 6, local router ID is 2.2.2.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.42.0.0/16     0.0.0.0                              32768 i

Total number of prefixes 1
```

To Prefer Inbound Traffic to Enter the Network Via ISP-B

1. Create a route-map on R1 that will prepend the local AS number to the AS-path three times:

```
R1#configure terminal
R1(config)#route-map PREFER_ISP2_INBOUND permit 10
R1(config-route-map)# set as-path prepend 65001 65001 65001
R1(config-route-map)#exit
```

2. Apply the route-map to the BGP configuration on R1:

```
R1(config)#router bgp 65001
R1(config-router)# neighbor 172.16.0.1 route-map PREFER_ISP2_INBOUND out
R1(config-router)#end
R1#
```

To Verify the AS-path Prepend Configuration is Being Applied

NOTE Unfortunately in IOS there is no good way to verify that the AS-path is being modified correctly. You will have to check the BGP table on the ISP router.

```
ISPA#show ip bgp
BGP table version is 8, local router ID is 172.16.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.42.0.0/16    172.16.0.2           0          0 65001 65001 65001 65001 i
```

To Prefer Outbound Traffic to Leave the Network Via ISP-B

1. Configure R2 to increase the local preference of all routes learned from ISP-B. :

```
R2#configure terminal
R2(config)#route-map PREFER_ISPB_OUTBOUND permit 10
R2(config-route-map)#set local-preference 110
R2(config-route-map)#router bgp 65001
R2(config-router)#neighbor 172.18.0.1 route-map PREFER_ISPB_OUTBOUND in
R2(config-router)#end
R2#
```

To Verify the Local Preference Value Has Been Changed

```
R2#show ip bgp | include Network|172.18.0.1
   Network          Next Hop            Metric LocPrf Weight Path
*  0.0.0.0          172.18.0.1               0    110      0 222 i
```

Junos Configuration

Now let's configure the same set of criteria in Junos. For IOS Engineers, watch how Junos relies upon policy configuration for most routing manipulation for both IGPs and BGP.

Once you're finished with this section, examine your final configuation. You'll notice how logical and well-organized it is. Further configuration is predictable, because Junos configuration follows a pretty standard format for configuring any objective.

To Configure the Junos Routers for Initial Connectivity

1. Set IP addresses on the interfaces on R1:

```
cjones@R1> configure
[edit]
cjones@R1# set interfaces ge-0/0/0 unit 0 family inet address 172.16.0.2/30

[edit]
cjones@R1# set interfaces ge-0/0/1 unit 0 family inet address 10.42.0.1/24

[edit]
cjones@R1# set interfaces lo0 unit 0 family inet address 1.1.1.1/32

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
cjones@R1>
```

2. Verify IP addresses on R1:

```
cjones@R1> show configuration interfaces
ge-0/0/0 {
```

```
    unit 0 {
        family inet {
            address 172.6.0.2/30;
        }
    }
}
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.42.0.1/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 1.1.1.1/24;
        }
    }
}
```

3. Set IP addresses on the interfaces on R2:

```
cjones@R2> configure
[edit]
cjones@R2# set interfaces ge-0/0/0 unit 0 family inet address 172.18.0.2/30

[edit]
cjones@R2# set interfaces ge-0/0/1 unit 0 family inet address 10.42.0.2/24

[edit]
cjones@R2# set interfaces lo0 unit 0 family inet address 2.2.2.2/32

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
cjones@R2>
```

4. Verify IP addresses on R2:

```
cjones@R2> show configuration interfaces
ge-0/0/0 {
    unit 0 {
        family inet {
            address 172.18.0.2/30;
        }
    }
}
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.42.0.2/24;
        }
    }
}
```

```
lo0 {
    unit 0 {
        family inet {
            address 2.2.2.2/24;
        }
    }
}
```

5. Set IP addresses on the interfaces on R3:

```
cjones@R3> configure
[edit]
cjones@R3# set interfaces ge-0/0/1 unit 0 family inet address 10.42.0.3/24

[edit]
cjones@R3# set interfaces lo0 unit 0 family inet address 3.3.3.3/32

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
cjones@R3>
```

6. Verify IP addresses on R3:

```
cjones@R3> show configuration interfaces
ge-0/0/1 {
    unit 0 {
        family inet {
            address 10.42.0.3/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 3.3.3.3/32;
        }
    }
}
```

To Verify Initial Connectivity

1. Ping R1 to R2:

```
cjones@R1> ping 10.42.0.2 rapid
PING 10.42.0.2 (10.42.0.2): 56 data bytes
!!!!!
--- 10.42.0.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.238/2.791/4.261/0.752 ms
```

2. Ping from R1 to R3:

```
cjones@R1> ping 10.42.0.3 rapid
PING 10.42.0.3 (10.42.0.3): 56 data bytes
```

```
!!!!!
--- 10.42.0.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.332/5.504/17.629/6.063 ms
```

3. Ping from R2 to R3:

```
cjones@R2> ping 10.42.0.3 rapid
PING 10.42.0.3 (10.42.0.3): 56 data bytes
!!!!!
--- 10.42.0.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.042/7.932/30.717/11.394 ms
```

4. Ping from R1 to ISP-A:

```
cjones@R1> ping 172.16.0.1 rapid
PING 172.16.0.1 (172.16.0.1): 56 data bytes
!!!!!
--- 172.16.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.625/7.827/31.212/10.268 ms
```

5. Ping from R2 to ISP-B:

```
cjones@R2> ping 172.18.0.1 rapid
PING 172.18.0.1 (172.18.0.1): 56 data bytes
!!!!!
--- 172.18.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.766/6.983/29.472/11.121 ms
```

To Configure the Junos Routers for OSPF

1. Let's now configure interfaces in OSPF area 0 on R1, configuring the loopback interface as passive and manually setting the OSPF router-id:

```
cjones@R1> configure
[edit]
cjones@R1# set protocols ospf area 0 interface ge-0/0/1.0

[edit]
cjones@R1# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R1# set routing-options router-id 1.1.1.1

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
cjones@R1>
```

2. Verify OSPF configuration on R1:

```
cjones@R1> show configuration routing-options
router-id 1.1.1.1;

cjones@R1> show configuration protocols
ospf {
    area 0.0.0.0 {
        interface ge-0/0/1.0;
        interface lo0.0 {
            passive;
        }
    }
}
```

3. Now let's do the same for R2 interfaces in OSPF area 0 on R2, configuring the loopback interface as passive, and manually setting the OSPF router-id:

```
cjones@R2> configure
[edit]
cjones@R2# set protocols ospf area 0 interface ge-0/0/1.0

[edit]
cjones@R2# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R2# set routing-options router-id 2.2.2.2

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
cjones@R2>
```

4. Verify OSPF configuration on R2:

```
cjones@R2> show configuration routing-options
router-id 2.2.2.2;

cjones@R2> show configuration protocols
ospf {
    area 0.0.0.0 {
        interface ge-0/0/1.0;
        interface lo0.0 {
            passive;
        }
    }
}
```

5. And the same for R3. Configure interfaces in OSPF area 0 on R3, configuring the loopback interface as passive, and manually setting the OSPF router-id:

```
cjones@R3> configure
[edit]
```

```
cjones@R3# set protocols ospf area 0 interface ge-0/0/1.0

[edit]
cjones@R3# set protocols ospf area 0 interface lo0.0 passive

[edit]
cjones@R3# set routing-options router-id 3.3.3.3

[edit]
cjones@R3# commit and-quit
commit complete
Exiting configuration mode
cjones@R3>
```

6. Verify OSPF configuration on R3:

```
cjones@R3> show configuration routing-options
router-id 3.3.3.3;

cjones@R3> show configuration protocols
ospf {
    area 0.0.0.0 {
        interface ge-0/0/1.0;
        interface lo0.0 {
            passive;
        }
    }
}
```

To Verify OSPF Configuration

1. Verify OSPF neighbor adjacencies on R1:

```
cjones@R1> show ospf neighbor
Address      Interface      State     ID        Pri  Dead
10.42.0.3    ge-0/0/1.0     Full      3.3.3.3   128    30
10.42.0.2    ge-0/0/1.0     Full      2.2.2.2   128    37
```

2. Verify OSPF neighbor adjacencies on R2:

```
cjones@R2> show ospf neighbor
Address      Interface      State     ID        Pri  Dead
10.42.0.1    ge-0/0/1.0     Full      1.1.1.1   128    32
10.42.0.3    ge-0/0/1.0     Full      3.3.3.3   128    30
```

3. Verify OSPF neighbor adjacencies on R3:

```
cjones@R3> show ospf neighbor
Address      Interface      State     ID        Pri  Dead
10.42.0.1    ge-0/0/1.0     Full      1.1.1.1   128    32
10.42.0.2    ge-0/0/1.0     Full      2.2.2.2   128    37
```

To Configure R1 and R2 to Inject a Default Route Into OSPF in Junos

1. Configure a static default route on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options static route 0/0 discard
```

2. Configure an export policy on R1:

```
[edit]
cjones@R1# edit policy-options policy-statement DEFAULT_TO_OSPF

[edit policy-options policy-statement DEFAULT_TO_OSPF]
cjones@R1# set from protocol static

[edit policy-options policy-statement DEFAULT_TO_OSPF]
cjones@R1# set from route-filter 0/0 exact

[edit policy-options policy-statement DEFAULT_TO_OSPF]
cjones@R1# set then accept

[edit policy-options policy-statement DEFAULT_TO_OSPF]
cjones@R1# top

[edit]
cjones@R1# show policy-options
policy-statement DEFAULT_TO_OSPF {
    from {
        protocol static;
        route-filter 0.0.0.0/0 exact;
    }
    then accept;
}
```

3. Configure the export policy on R1 to inject a default route into the OSPF process:

```
[edit]
cjones@R1# set protocols ospf export DEFAULT_TO_OSPF

[edit]
cjones@R1# show protocols ospf
export DEFAULT_TO_OSPF;
area 0.0.0.0 {
    interface ge-0/0/1.0;
    interface lo0.0 {
        passive;
    }
}
```

4. Verify the configuration on R1, and commit:

```
[edit]
cjones@R1# show | compare
```

```
[edit routing-options]
+   static {
+       route 0.0.0.0/0 discard;
+   }
[edit protocols ospf]
+   export DEFAULT_TO_OSPF;
[edit]
+  policy-options {
+      policy-statement DEFAULT_TO_OSPF {
+          from {
+              protocol static;
+              route-filter 0.0.0.0/0 exact;
+          }
+          then accept;
+      }
+  }

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

5. Next, we need to add the exact same configuration to R2:

Let's try something new here. Instead of re-typing everything, let's configure R2 using the patch syntax in step 4. Copy the `compare` output from above and use the `load patch terminal` command on R2. Paste, and hit ctrl+d to finish. Then commit the changes, like this:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# load patch terminal
[Type ^D at a new line to end input]
[edit routing-options]
+   static {
+       route 0.0.0.0/0 discard;
+   }
[edit protocols ospf]
+   export DEFAULT_TO_OSPF;
[edit]
+  policy-options {
+      policy-statement DEFAULT_TO_OSPF {
+          from {
+              protocol static;
+              route-filter 0.0.0.0/0 exact;
+          }
+          then accept;
+      }
+  }|
load complete

[edit]
```

```
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

To Verify the OSPF Default Route in Junos

1. Check for the OSPF type-5 LSA in the OSPF database on R3:

```
cjones@R3> show ospf database external
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern   0.0.0.0          1.1.1.1          0x80000001   599  0x22 0xe2cb  36
Extern   0.0.0.0          2.2.2.2          0x80000001   220  0x22 0xc4e5  36
```

Check for the OSPF external route in the RIB on R3:

NOTE The exact keyword is used in this case because otherwise our entire RIB will be shown, since all routes match 0.0.0.0/0.

```
cjones@R3> show route 0.0.0.0/0 exact

inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[OSPF/150] 00:04:35, metric 0, tag 0
                      to 10.42.0.1 via ge-0/0/1.0
                    > to 10.42.0.2 via ge-0/0/1.0
```

To Configure Junos Routers R1 and R2 for EBGP to ISP-A and ISP-B

1. Configure R1 (AS 65001) to peer with ISP-A (AS 111):

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options autonomous-system 65001

[edit]
cjones@R1# set protocols bgp group ISP-A type external neighbor 172.16.0.1 peer-
as 111
```

2. Verify the BGP configuration on R1:

```
[edit]
cjones@R1# show routing-options autonomous-system
65001;

[edit]
cjones@R1# show protocols bgp
group ISP-A {
    type external;
    neighbor 172.16.0.1 {
        peer-as 111;
    }
}
```

3. Commit the BGP configuration on R1:

```
[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

4. Configure R2 (AS 65001) to peer with ISP-B (AS 222):

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set routing-options autonomous-system 65001

[edit]
cjones@R2# set protocols bgp group ISP-B type external neighbor 172.18.0.1 peer-
as 222
```

5. Verify the BGP configuration on R2:

```
[edit]
cjones@R2# show routing-options autonomous-system
65001;

[edit]
cjones@R2# show protocols bgp
group ISP-B {
    type external;
    neighbor 172.18.0.1 {
        peer-as 222;
    }
}
```

6. Commit the BGP configuration on R2:

```
[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

To Verify EBGP Configuration on IOS

1. Verify R1's EBGP peering with ISP-A:

```
cjones@R1> show bgp neighbor 172.16.0.1
Peer: 172.16.0.1+57730 AS 111  Local: 172.16.0.2+179 AS 65001
  Type: External    State: Established    Flags: <ImportEval Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference PeerAS Refresh>
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 172.16.0.1      Local ID: 1.1.1.1          Active Holdtime: 90
  Keepalive Interval: 30         Peer index: 0
  BFD: disabled, down
  Local Interface: ge-0/0/0.0
```

2. Verify R1 is receiving the 111.111.111.0/24 route advertised by ISP-A by inspecting the Adj-RIB-In table on R1:

```
cjones@R1> show route receive-protocol bgp 172.16.0.1

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
  Prefix     Nexthop       MED     Lclpref    AS path
* 111.111.111.0/24        172.16.0.1                               111 I
```

3. Verify the BGP-learned route from ISP-A is being installed in the RIB on R1:

```
cjones@R1> show route protocol bgp

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:01:30, localpref 100
                      AS path: 111 I
                    > to 172.16.0.1 via ge-0/0/0.0
```

4. Verify R2's EBGP peering with ISP-B:

```
cjones@R2> show bgp neighbor 172.18.0.1
Peer: 172.18.0.1+179 AS 222    Local: 172.18.0.2+56620 AS 65001
  Type: External    State: Established    Flags: <ImportEval Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Options: <Preference PeerAS Refresh>
  Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 172.18.0.1      Local ID: 2.2.2.2          Active Holdtime: 90
  Keepalive Interval: 30         Peer index: 0
  BFD: disabled, down
  Local Interface: ge-0/0/0.0
```

5. Verify R2 is receiving the 111.111.111.0/24 route advertised by ISP-B by inspecting the Adj-RIB-In table on R2:

```
cjones@R2> show route receive-protocol bgp 172.18.0.1

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
  Prefix     Nexthop       MED     Lclpref    AS path
* 111.111.111.0/24        172.18.0.1                               222 I
```

6. Verify the BGP-learned route from ISP-B is being installed in the RIB on R2:

```
cjones@R2> show route protocol bgp

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:02:42, localpref 100
                      AS path: 222 I
                    > to 172.18.0.1 via ge-0/0/0.0
```

To Configure Junos Routers R1 and R2 for IBGP

1. Configure R1 for IBGP to R2, using the loopback interface for peering:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set protocols bgp group IBGP type internal neighbor 2.2.2.2 local-
address 1.1.1.1 peer-as 65001
```

2. Configure a static route on R1 to enable full reachability to R2's loopback:

Since OSPF is running, this step isn't necessary, but let's configure it anyway for demonstration's sake. Since it is preferred to use the OSPF route, give the static route a high route preference. The only time the floating static route will ever get used is in the event of an IGP failure.

```
[edit]
cjones@R1# set routing-options static route 2.2.2.2/32 next-
hop 10.42.0.2 preference 254
```

3. Configure a policy on R1 to change the next-hop for all IBGP prefixes sent to R2 to its own interface address:

```
[edit]
cjones@R1# set policy-options policy-statement NHS then next-hop self
```

4. Configure R1 to use the *NHS* policy on its IBGP peering to R2:

```
[edit]
cjones@R1# set protocols bgp group IBGP export NHS
```

5. Verify the changes to R1, and commit the configuration:

```
[edit]
cjones@R1# show | compare
[edit routing-options static]
     route 0.0.0.0/0 { ... }
+    route 2.2.2.2/32 {
+        next-hop 10.42.0.2;
+        preference 254;
+    }
[edit protocols bgp]
     group ISP-A { ... }
+    group IBGP {
+        type internal;
+        export NHS;
+        neighbor 2.2.2.2 {
+            local-address 1.1.1.1;
+            peer-as 65001;
+        }
+    }
[edit policy-options]
+   policy-statement NHS {
```

```
+       then {
+           next-hop self;
+       }
+   }

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

6. Configure R2 for IBGP to R1, using the loopback interface for peering:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set protocols bgp group IBGP type internal neighbor 1.1.1.1 local-
address 2.2.2.2 peer-as 65001
```

7. Configure a static route on R2 to enable full reachability to R1's loopback:

```
[edit]
cjones@R2# set routing-options static route 1.1.1.1/32 next-
hop 10.42.0.1 preference 254
```

8. Configure a policy on R2 to change the next-hop for all IBGP prefixes sent to R1 to its own interface address:

```
[edit]
cjones@R2# set policy-options policy-statement NHS then next-hop self
```

9. Configure R2 to use the *NHS* policy on its IBGP peering to R1:

```
[edit]
cjones@R2# set protocols bgp group IBGP export NHS
```

10. Verify the changes to R1, and commit the configuration:

```
[edit]
cjones@R2# show | compare
[edit routing-options static]
     route 0.0.0.0/0 { ... }
+    route 1.1.1.1/32 {
+        next-hop 10.42.0.1;
+        preference 254;
+    }
[edit protocols bgp]
     group ISP-B { ... }
+    group IBGP {
+        type internal;
+        export NHS;
+        neighbor 1.1.1.1 {
+            local-address 2.2.2.2;
```

```
+            peer-as 65001;
+        }
+    }
[edit policy-options]
+   policy-statement NHS {
+       then {
+           next-hop self;
+       }
+   }

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

To Verify IBGP Configuration Between R1 and R2

1. Verify the IBGP adjacency between R1 and R2:

```
cjones@R1> show bgp neighbor 2.2.2.2
Peer: 2.2.2.2+63702 AS 65001   Local: 1.1.1.1+179 AS 65001
  Type: Internal    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: None
  Export: [ NHS ]
  Options: <Preference LocalAddress PeerAS Refresh>
  Local Address: 1.1.1.1 Holdtime: 90 Preference: 170
  Number of flaps: 0
  Peer ID: 2.2.2.2         Local ID: 1.1.1.1          Active Holdtime: 90
```

2. Verify the R1 is receiving prefixes from R2 with the correct next-hop:

```
cjones@R1> show route receive-protocol bgp 2.2.2.2

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
  Prefix     Nexthop       MED     Lclpref    AS path
* 111.111.111.0/24        2.2.2.2                      100        222 I
```

3. Verify the prefix from R2 is being installed in the RIB on R1:

```
cjones@R1> show route protocol bgp 111.111.111.0/24

inet.0: 10 destinations, 13 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:21:51, localpref 100
                      AS path: 111 I
                    > to 172.16.0.1 via ge-0/0/0.0
                    [BGP/170] 00:02:18, localpref 100, from 2.2.2.2
                      AS path: 222 I
                    > to 10.42.0.2 via ge-0/0/1.0
```

4. Verify the R2 is receiving prefixes from R1 with the correct next-hop:

```
cjones@R2> show route receive-protocol bgp 1.1.1.1

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
  Prefix     Nexthop       MED     Lclpref    AS path
* 111.111.111.0/24        1.1.1.1                      100        111 I
```

5. Verify the prefix from R1 is being installed in the RIB on R2:

```
cjones@R2> show route protocol bgp 111.111.111.0/24

inet.0: 10 destinations, 13 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:03:28, localpref 100
                      AS path: 222 I
                    > to 172.18.0.1 via ge-0/0/0.0
                    [BGP/170] 00:23:01, localpref 100, from 1.1.1.1
                      AS path: 111 I
                    > to 10.42.0.1 via ge-0/0/1.0
```

To Advertise the Aggregate Prefix 10.42.0.0/16 to AS 111 and AS 222

1. Advertsing an aggregate is fairly simple in Junos. While it requires more configuration than in IOS, it follows the same policy structure you should be fairly used to seeing by now.

In order to advertise an aggregate, you must first create it. This is done using nearly the same syntax as a static route. Once created, match that aggregate in a policy, and apply that policy as an export policy in BGP.

It is important to note that a reject term is required at the bottom of the policy to ensure that BGP does not advertise anything else besides the aggregate.

1. Create the aggregate on R1:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set routing-options aggregate route 10.42.0.0/16
```

2. Configure a policy on R1 that matches the aggregate and accepts it, but rejects everything else:

```
[edit]
cjones@R1# edit policy-options policy-statement AGG_TO_ISP

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R1# set term ACCEPT_AGG from protocol aggregate
```

```
[edit policy-options policy-statement AGG_TO_ISP]
cjones@R1# set term ACCEPT_AGG from route-filter 10.42.0.0/16 exact

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R1# set term ACCEPT_AGG then accept

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R1# set term REJECT_OTHERS then reject

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R1# top
```

3. Configure the policy under the export statement under the group or neighbor in the BGP configuration:

```
[edit]
cjones@R1# set protocols bgp group ISP-A neighbor 172.16.0.1 export AGG_TO_ISP
```

4. Verify the changes on R1 and commit:

```
[edit]
cjones@R1# show | compare
[edit routing-options]
+   aggregate {
+       route 10.42.0.0/16;
+   }
[edit protocols bgp group ISP-A neighbor 172.16.0.1]
+     export AGG_TO_ISP;
[edit policy-options]
+   policy-statement AGG_TO_ISP {
+       term ACCEPT_AGG {
+           from {
+               protocol aggregate;
+               route-filter 10.42.0.0/16 exact;
+           }
+           then accept;
+       }
+       term REJECT_OTHERS {
+           then reject;
+       }
+   }

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

5. Create the aggregate on R2:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set routing-options aggregate route 10.42.0.0/16
```

6. Configure a policy on R2 that matches the aggregate and accepts it, but rejects everything else. Note that there is an implicit accept if a reject statement is not configured.

```
[edit]
cjones@R2# edit policy-options policy-statement AGG_TO_ISP

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R2# set term ACCEPT_AGG from protocol aggregate

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R2# set term ACCEPT_AGG from route-filter 10.42.0.0/16 exact

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R2# set term ACCEPT_AGG then accept

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R2# set term REJECT_OTHERS then reject

[edit policy-options policy-statement AGG_TO_ISP]
cjones@R2# top
```

7. Configure the policy under the export statement under the group or neighbor in the BGP configuration:

```
[edit]
cjones@R2# set protocols bgp group ISP-B neighbor 172.18.0.1 export AGG_TO_ISP
```

8. Verify the changes on R2 and commit:

```
[edit]
cjones@R2# show | compare
[edit routing-options]
+   aggregate {
+       route 10.42.0.0/16;
+   }
[edit protocols bgp group ISP-B neighbor 172.18.0.1]
+     export AGG_TO_ISP;
[edit policy-options]
+   policy-statement AGG_TO_ISP {
+       term ACCEPT_AGG {
+           from {
+               protocol aggregate;
+               route-filter 10.42.0.0/16 exact;
+           }
+           then accept;
+       }
+       term REJECT_OTHERS {
+           then reject;
+       }
+   }

[edit]
cjones@R2# commit and-quit
```

```
commit complete
Exiting configuration mode
```

To Verify if the Aggregate Prefix is Being Sent to AS 111 and AS 222

1. Check the Adj-RIB-Out table on R1:

```
cjones@R1> show route advertising-protocol bgp 172.16.0.1

inet.0: 11 destinations, 14 routes (11 active, 0 holddown, 0 hidden)
  Prefix      Nexthop       MED     Lclpref    AS path
* 10.42.0.0/16            Self                                  I
```

2. Check the Adj-RIB-Out table on R2:

```
cjones@R2> show route advertising-protocol bgp 172.18.0.1

inet.0: 11 destinations, 14 routes (11 active, 0 holddown, 0 hidden)
  Prefix      Nexthop       MED     Lclpref    AS path
* 10.42.0.0/16            Self                                  I
```

To Prefer That Inbound Traffic Enters the Network Via ISP-B

Making traffic enter the AS in Junos is done using AS path prepending. To accomplish this, you use the same policy structure we have come to know and love. In this case, you already have a policy exporting our aggregate to ISP-A, so you can simply add to that.

1. Modify the export policy on R1 to prepend the AS:

```
cjones@R1> configure
Entering configuration mode

[edit]
cjones@R1# set policy-options policy-statement AGG_TO_ISP term ACCEPT_AGG then as-
path-prepend "65001 65001"

[edit]
cjones@R1# commit and-quit
commit complete
Exiting configuration mode
```

To Verify the AS-path Prepend Configuration is Being Applied

In Junos, you can simply look at the Adj-RIB-Out table to see your modified AS path. This table shows the changes to your BGP-advertised prefixes after policy has been applied.

1. Check the Adj-RIB-Out table:

```
cjones@R1> show route advertising-protocol bgp 172.16.0.1

inet.0: 11 destinations, 14 routes (11 active, 0 holddown, 0 hidden)
  Prefix      Nexthop       MED     Lclpref    AS path
* 10.42.0.0/16           Self                                  65001 65001 [65001] I
```

To Prefer That Outbound Traffic Leaves the Network Via ISP-B

1. Once again you can use policy to modify the prefixes learned via BGP: in this case, a simple import policy on R2 that increases the local preference of all routes from ISP-B can be used. Create a policy on R2 that increases the local preference of all routes learned from ISP-B:

```
cjones@R2> configure
Entering configuration mode

[edit]
cjones@R2# set policy-options policy-statement ISPB-LOCALPREF then local-
preference 110
```

2. Apply the policy as import under the BGP group or neighbor on R2. Then commit:

```
[edit]
cjones@R2# set protocols bgp group ISP-B neighbor 172.18.0.1 import ISPB-LOCALPREF

[edit]
cjones@R2# commit and-quit
commit complete
Exiting configuration mode
```

To Verify the Local Preference Value Has Been Changed

1. Check the RIB on R2. Notice that the prefix is no longer being learned from R1. This is because R1's best route to the prefix is now learned via R2, and R1 will not re-advertise the route back to the source it learned it from originally.

```
cjones@R2> show route protocol bgp

inet.0: 11 destinations, 13 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:51:34, localpref 110
                      AS path: 222 I
                    > to 172.18.0.1 via ge-0/0/0.0
```

2. Check the RIB on R1. Make sure the preferred path is now out towards ISP-B via R2:

```
cjones@R1> show route protocol bgp

inet.0: 11 destinations, 14 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

111.111.111.0/24   *[BGP/170] 00:02:42, localpref 110, from 2.2.2.2
                      AS path: 222 I
                    > to 10.42.0.2 via ge-0/0/1.0
                    [BGP/170] 01:09:57, localpref 100
                      AS path: 111 I
                    > to 172.16.0.1 via ge-0/0/0.0
```

Conclusion

In this chapter you have learned how to configure and compare a simple network configuration sharing common elements between both IOS and Junos. It is clear that Junos relies heavily upon policy configuration for most routing manipulation for both IGPs and BGP, but as you can see there's an elegance in the hierarchical makeup of Junos. It isn't C Programming. It isn't harder or more complex than IOS. It's just more powerful.

Tasks you have learned here can be applied across any Junos device, including routers, switches, firewalls, and more. And the resources at the end of this book can help.

This book began comparing tasks, then moved to configurations, and then to configuring a sample topology. You should now be familiar enough working with Junos to jump right in. The networking science doesn't change when moving to Junos, it just gets a little easier.

What to Do Next & Where to Go ...

http://www.juniper.net/dayone

The free PDF of this book is staged here for download. You should also check out the other free *Day One* books on Junos, including:

- *Day One: Exploring the Junos CLI*
- *Day One: Configuring Junos Basics*
- *Day One: Junos Tips, Techniques, and Templates*
- *This Week: Hardening Junos Devices*

http://forums.juniper.net/t5/IOS-to-Junos-I2J-Tips-Contest/bd-p/I2JTips

The Juniper-sponsored J-Net Communities forum is dedicated to sharing information, best practices, and questions about Juniper products, technologies, and solutions. Here, check out the dozens of IOS to Junos contest entries for a I2J tips contest.

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

The Junos technical documentation includes everything you need to understand and configure all aspects of Junos

https://learningportal.juniper.net/juniper/user_activity_info.aspx?id=3310

Designed for network engineers who are familiar with Cisco's IOS, this course builds on existing IOS configuration knowledge to provide a high-level overview of Junos OS, how it works, and how it compares with IOS.

http://itunes.apple.com/us/podcast/junos-as-a-second-language/id276663160

And here's a Podcast of the same great educational course.