JUNIPER NETWORKS | Engineering Simplicity

DAY ONE: BEGINNER'S GUIDE TO LEARNING JUNOS®

By Martin Brown, Chris Parker, Yasmin Lara, Peter Klimai, Christian Scholz, Tom Dwyer, Paul Clarke, and Jeff Fry

DAY ONE: BEGINNER'S GUIDE TO LEARNING JUNOS®

The Juniper Ambassadors are a diverse set of independent network engineers, consultants, and architects who work in the field with Juniper technologies on a daily basis. Their mission is to spread the word about the power and scalability of Juniper's network offerings. Indeed, they are responsible for over a dozen Day One books and countless social media appearances, videos, and training sessions. But they have always wanted to fill a major gap in the networking cannon and create a true beginner's guide to the most powerful networking OS in the world. And now they have done it, the Ambassador way: hands on, to the point, and complete. It's all here in the *Beginner's Guide to Learning Junos*.

"Whether you are new to Juniper or not, there isn't a more complete introduction to Junos available. I wish this book was around when I started my network journey years ago. The Ambassador authors bring together so much knowledge and expert field experience that this book is a one-stop-shop to take you from beginner to Junos expert."

Melchior Aelmans, Lead Engineer Cloud Providers, Juniper Networks

"Nowadays there are many vendors offering similar services and its difficult to learn them all. What network engineers need is a clear and easy-to-follow guide to assist them in their learning journey. Any one new to Juniper will find this book an absolute must read – the authors have presented the topics in a clear and simplified way without losing any of the details."

Joy Horton, Service Desk Analyst, Xalient

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

- The purpose and function of devices in the Juniper portfolio, from routers to switches to firewalls.
- How the Junos CLI works, and the power it brings to your understanding of your network.
- How to use the Junos hierarchy to accurately predict the commands and configuration you'll need.
- To configure and troubleshoot Juniper hardware, including checking for physical interface errors.
- To configure and troubleshoot protocols that run on Juniper routers, including log analysis, debugging, ping, traceroute, and commands to let you monitor control-plane traffic in and out of an interface.
- How routing works within Junos, not only on a default protocol level, but in terms of your ability to manipulate those default behaviors.

ISBN 978-1-941441-77-0
53500
9 781941 441770

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

JUNIPER NETWORKS

Day One: Beginner's Guide to Junos®

by Martin Brown, Chris Parker, Yasmin Lara, Peter Klimai, Christian Scholz, Tom Dwyer, Paul Clarke, Jeff Fry

Chapter 1: Junos Fundamentals 15

Chapter 2: User Interface 42

Chapter 3: Junos Configuration Basics 119

Chapter 4: Operational Monitoring and Maintenance 147

Chapter 5: Routing Fundamentals 210

Chapter 6: Routing Policies 231

Chapter 7: Troubleshooting Policies 296

Chapter 8:: Firewall Filter Concepts 316

Appendix: How to Build Your Own Virtual Lab 346

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Martin Brown, Chris Parker, Yasmin Lara, Peter Klimai, Christian Scholz, Tom Dwyer, Paul Clarke, Jeff Fry
Technical Reviewers: Nick Ryce, Victor Gonzalez, Joy Horton, Ben Dale
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
TechLibrary Resources: Saheer Karimbayil
J-Net Resources: Steve Puluka

Printed in the USA by Vervante Corporation.
Version History: v1, August, 2020
2 3 4 5 6 7 8 9 10

Comments, errata: dayone@juniper.net

Welcome to Day One
This book is part of the *Day One* library, produced and published by Juniper Networks Books. *Day One* books cover the Junos OS and Juniper Networks network administration with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

Download free PDF editions at http://www.juniper.net/dayone

Available on the Juniper app: Junos Genius

Purchase the paper edition at Vervante Corporation (www.vervante.com).

About the Authors

Martin Brown (Chapter 1) is a Senior Solutions Consultant for a cyber security specialist based in the UK and has been a Juniper Ambassador since 2012. Martin started his career in IT over 20 years ago supporting Macintosh computers, and in 1999 earned his first certification by becoming an MCP then an MCSE. In the past nine years he has progressed to networking, implementing, and supporting network devices in a number of different environments including airports, retail, warehouses and service providers. His knowledge covers a broad range of network device types and network equipment from most of the major vendors including Cisco, F5, Checkpoint, Palo Alto, and of course, Juniper.

Chris Parker (Chapter 2 and Appendix) is a British network engineer, though he prefers to think of himself as a citizen of the Internet. He has worked in the Service Provider sector for over 10 years and has a particular passion for BGP, MPLS, and IS-IS. He holds six Juniper certifications and is currently working towards the JNCIE-SP. He blogs at NetworkFunTimes.com, where he teaches Juniper technologies with a smile and a sense of humor. Find him on Twitter at @NetworkFunTimes.

Yasmin Lara (Chapter 6 and Chapter 7) Yasmin Lara (Chapters 6 and 7) is a 4xJNCIE, JNCIP-CLOUD, CCNP-SP, and a Juniper Ambassador. She has 20+ years of networking experience as an Instructor, Network Engineer and Consultant. Currently, she works for Cypress Consulting, as a Senior Network Consultant for a major Service Provider in the US. Her biggest professional passion is to understand and learn about technology and figure out the best way to explain it to others. She is a baseball (go Nationals!!) and Star Wars fan and a Third Degree Black Belt in Tae Kwon Do. In her spare time she enjoys spending time with her husband and two teenage boys. Yasmin wants to thank her fellow Ambassadors for the opportunity of being part of this project and her husband for all his support.

Peter Klimai (Chapter 8 and Chapter 3) is a Juniper Ambassador working as Lead Engineer and Instructor at Poplar Systems (a Juniper Partner in Russia since 2001). He is certified JNCIE-SEC #98, JNCIE-ENT #393, JNCIE-SP #2253, JNCIP-DC, JNCIP-Cloud, and JNCI, and has several years of experience supporting Juniper equipment for small and large companies. Peter teaches Juniper classes on routing, security, automation and troubleshooting, and is especially enthusiastic about network automation using various tools, as well as network function virtualization. He is also an author of *Day One: Automating Junos with Salt* and co-author of several other *Day One* books.

Christian Scholz (Chapter 3 and Appendix), @chsjuniper, is a German Senior Consultant for Network and Security, and a Juniper Ambassador, who is currently working for Telonic, a System Integrator based in Cologne. Christian holds 17 Juniper Certifications, including his JNCIE-SEC #374, and has over a decade of experience. He enjoys blogging and giving knowledge back to the network community, especially with topics about building labs, automation, and IPv6.

Tom Dwyer (Chapter 5) is the Director of Network Infrastructure at Enverus, the leading energy SaaS company delivering highly-technical insights and predictive/prescriptive analytics. He has over 20 years of experience focused on networking, security, and data center technologies. Tom is a Juniper Ambassador and is certified by Juniper as a JNCIE-ENT #424.

Paul Clarke (Chapter 3) is a Customer Solutions Architect working for Fujitsu in the UK and is a Juniper Ambassador. He specializes in next generation networks such as Cloud, SD-WAN, Contrail and Mist AI. He has over 20 years of experience focused on networking and holds five JNCIP certifications.

Jeff Fry (Chapter 4) current works as a Senior Technical Consultant at a global VAR. His main focus is working with customers on their Juniper deployments, migrations from other vendors to Juniper, or assisting them with improving their existing environments though better leveraging of existing hardware.

Technical Reviewers

Nick Ryce (Technical Editor) is a Senior Network Architect for Commsworld, an ISP based in Scotland and is a Juniper Ambassador. Nick has over a decade of experience working within the Service Provider industry and has worked with a variety of vendors including Cisco, Nortel, HP, and Juniper. Nick is currently certified as JNCIE-ENT #232.

Victor Gonzalez (Technical Editor) is a Senior Network Architect for Atlantic Metro, a cloud hosting, Global Network Connectivity Provider, and a Juniper Ambassador. Victor has twenty years of experience working within Enterprise and Service Provider networks. Well versed in multi-vendor hardware and platforms such as, Cisco, Brocade, Extreme, Arista, and Juniper, Victor currently holds JNCIEENT #00510.

Authors' Acknowledgments

Martin Brown thanks Julie Wider for the creation of the Juniper Ambassador program and all the work she did and the support she gave the Ambassadors. I'd also like to thank 'Salem' just for being there when I needed him and for looking after me when I needed company and inspiration.

Chris Parker would like to thank Mike Noble and Georgia Carter for reading early versions of my chapter. I would also like to give a huge thank-you to Julie Wider for all the support she's given me during my time in the Ambassador program; Patrick Ames for his support and patience during my many re-edits; and John Goodman for making great movies.

Peter Klimai would like to thank Julie Wider, Sarah Lesway-Ball, Patrick Ames, Leonid Mirenkov, Alex Tarkhov, and his family.

Yasmin Lara Yasmin wants to thank her fellow Ambassadors for the opportunity of being part of this project and her husband for all his support.

Christian Scholz would like to thank Svenja Scholz, Julie Wider, Staci Robinson, and Adam Rabidoux: without your help, I would not nearly be where I am currently. I can't thank you enough!

Tom Dwyer I would like to thank my family for their love, encouragement and support. Thank you to Victor Gonzales and Nick Ryce for their technical review. Thank you to Patrick Ames for the amazing work putting these Day One books together. Finally I would like to thank Julie Wider, her amazing work behind the scenes helped to build the Juniper Ambassador program into what it is today.

Paul Clarke would like to thank Martin Brown and Chris Parker.

Jeff Fry would like to thank Julie Wider for all the motivate that she has given me over the years. I would also like to thank my family and friends for understanding why I was pecking at the keys for all those nights. Lastly, I want to thank you the reader for being willing to spend your time reading our words.

Assumptions About Readers

The authors assume that readers of this book should be familiar with the following:

- The seven-layer OSI model and the basics of networking
- The difference between a router, switch, and firewall
- Networking terminology such as SNMP, SMTP, ICMP, TCP, and UDP

What You Need to Know Before Reading This Book

Commands used in this book and associated output were both run and taken from a mixture of physical and virtual devices in order to give an accurate representation of what you would see in the real world, so:

- You need a good understanding of networking concepts such as how routers make routing decisions.
- You should ideally know the differences between TCP and UDP and understand the three-way handshake.
- Real world network experience from any vendor would be advantageous.
- How to open putty or a Mac terminal in order to manage network devices.

After Reading This Book You'll Be Able To

- Understand the purpose and function of the key devices in the Juniper product set, from routers to switches to firewalls.
- Understand how the Junos command-line works, and the power it brings to your understanding of your network.
- Appreciate how to use the hierarchy of Junos to accurately predict the commands and configuration you'll need.
- Configure and troubleshoot Juniper hardware, including checking for physical interface errors.
- Configure and troubleshoot protocols that run on Juniper routers, including log analysis, debugging, ping, traceroute, and commands to let you monitor control-plane traffic in and out of an interface.
- Explain how routing works within Junos, not only on a default protocol level, but in terms of your ability to manipulate those default behaviors.

How This Book Is Set Up

Chapter 1 provides an introduction to the Juniper product set. We explain the difference between three key products in Juniper's hardware line: an EX switch, an SRX firewall, and an MX router. We also look at the PTX and ACX series. We'll then look at the architecture of a typical Juniper device, specifically the difference between the control plane and forwarding plane. We explain what these are and why keeping them separate can give you many advantages in terms of stability. We round off with a look at some of the software daemons that power the box behind the scenes.

Chapter 2 is where you'll learn how to actually log on to a Juniper device. We also introduce you to the Junos command-line interface (CLI). You'll learn the philosophy of the Junos CLI; you'll see the consistent and predictable hierarchy of commands; you'll learn some basic "show" commands to let you view the operational status of the box; you'll learn some simple configuration changes you can make; and you'll learn a variety of ways that you can save your configuration and roll back to previous configurations.

Chapter 3 takes the lessons around configuration to the next level. You'll learn how to create new user accounts, how to turn on SSH and set a root password, how to automatically send configurations to an archive server, and how to create your own custom log files. We'll also teach you how to use "traceoptions" to get some highly detailed debug output about the behavior of the box, to take your troubleshooting to the next level.

Chapter 4 focuses on monitoring and maintenance. We offer a variety of commands to view live information about the performance of both hardware and software, from checking licenses and hardware components to temperature and memory usage. We'll show you how to check alarm lights, "core dumps," system uptime, and interface problems. By the end of this chapter you'll feel confident in actually troubleshooting your Juniper box, and finding the cause of problems in production.

Chapter 5 brings routing fundamentals to the table. Here you'll learn about the particular way that Junos devices handle IP routes, or "prefixes" as they're known. You'll learn how different protocols have different route preferences, and the process by which a route is actually entered into both the routing table and the forwarding table. You'll also learn about routing instances, which offer a variety of different ways in which you can run one physical box to contain multiple virtual routers and switches inside it. We'll show you how to make static routes, and how to configure a basic OSPF setup.

Chapter 6 takes routing fundamentals to the next level, with the introduction of routing policies. You'll learn how you can use policies to control what things a protocol advertises out, and accepts in. You'll see that these policies can match IP routes based on a huge number of match conditions, and that Junos can then manipulate that prefix in almost any way you like before it comes in or goes out of the router. You'll also learn that each protocol has a different default behavior, and that understanding these default behaviors is crucial to routing success.

Chapter 7 builds on Chapter 6, and introduces a methodology for troubleshooting routing policies. It lays out some common mistakes and gotchas, such as incorrectly ordering your policies, and forgetting the correct default behavior of a protocol. By the end of this chapter, you will understand that the order of actions in your routing policy is critical to the correct behavior of your router.

Chapter 8 explains firewall filters. Some other vendors call these "access lists," but in this chapter you'll learn that Junos crafts such "access lists" in a much more human-readable way. In fact, students of Chapters 6 and 7 will immediately recognise the syntax format. In this chapter you'll learn how to write a firewall filter, what things you can match on, and what actions you can take. You'll find out how to log and count traffic, and how to easily use a firewall filter to "police" your traffic to a certain bandwidth of your choosing.

The Appendix will help you to set up your own virtual lab so you can get hands-on with Junos. We'll show you three different ways that you can build a lab on your own computer, completely for free, so that you can practice the CLI and create different topologies of your own.

Junos Resources in the Juniper TechLibrary

JUNOS BASICS	
Junos OS Documentation	https://www.juniper.net/documentation/product/en_US/junos-os#cat=product_documentation
Overview for Junos OS	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/junos-overview.html
Day 1 + Junos OS	https://www.juniper.net/documentation/en_US/day-one-plus/junos-os/id-step-1-begin.html
Getting Started Guide for Junos OS	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/getting-started.html
User Access and Authentication User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/user-access.html
CLI User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/junos-cli/junos-cli.html
CLI Explorer	https://apps.juniper.net/cli-explorer/
Software Installation and Upgrade Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/software-installation-and-upgrade/software-installation-and-upgrade.html
Introducing Junos OS Evolved	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/introducing-evo-guide.html

ROUTING POLICIES, PROTOCOLS, FIREWALL FILTERS, AND SERVICES	
Routing Protocols Overview	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-overview.html
Protocol-Independent Routing Properties User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-static-routing.html
Routing Policies, Firewall Filters, and Traffic Policers User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-policy/config-guide-policy.html
OSPF User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-ospf.html
BGP User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-bgp.html

RIP User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-rip.html
IS-IS User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-is-is.html
MPLS Applications User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-mpls-applications/config-guide-mpls-applications.html
Class of Service User Guide (Routers and EX9200 Switches)	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/cos/config-guide-cos.html
Class of Service User Guide (Security Devices)	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/security/cos-overview.html
Adaptive Services Interfaces User Guide for Routing Devices	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/services-interfaces/router-services-application-properties.html
Broadband Subscriber Services User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/subscriber-access/subscriber-mgmt-advanced-provisioning.html

ADDITIONAL RESOURCES	
Ethernet Switching User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/layer-2/layer2-multicast.html
Ethernet Interfaces User Guide for Routing Devices	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-network-interfaces/network-interfaces-ethernet.html
Interfaces Fundamentals for Routing Devices	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-network-interfaces/network-interfaces-fundamentals.html
Interfaces User Guide for Switches	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/qfx-series/ethernet-interfaces-switches.html
J-Web for SRX Series Documentation	https://www.juniper.net/documentation/product/en_US/j-web-srx-series
Chassis-Level User Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/system-basics/router-chassis.html
Network Management and Monitoring Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/network-management/network-management.html
REST API Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/rest-api/rest-api.html
NETCONF XML Management Protocol Developer Guide	https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/netconf-guide/netconf.html

J-Net: A Resource for Using and Learning Junos

The internet is a great place to find people with common interests and share experiences, and for Junos that community is J-Net. The J-Net community has the people and tools to help you become familiar and ultimately comfortable with using Junos implementations.

The Juniper Ambassadors encourage you to join the J-Net community and use the experience to build your Junos skills. Two other Ambassador-approved resources are the Open Learning Program webinars, and vLabs, with its online Junos lab access to practice.

Most of us only touch a small fraction of the features, technologies, and protocols that make the networking field as diverse and interesting as it is today, and J-Net can really help you get familiar with these direct implementations. And the community participation can expand your horizons with myriad other possibilities that the respected Junos OS offers.

Nothing is more frustrating than having unexpected network issues or systems not working as expected when you deploy them. We all have been in those situations. Sometimes you just can't see the tree for the forest and another set of eyes from J-Net can give you the quick fix. In some situations the details are tricky and the threads go on and on until a root cause or final configuration change is found to get packets flowing freely again.

The great benefit from these forum threads is that they'll make you a better engineer and you'll understand how all the Juniper technologies work together, and how to troubleshoot and deploy them in your own situation. There is nothing like an engineer that has solved a real issue on an actual network. It feels good to help. Real networks are solving real problems.

Welcome to Junos. The Ambassadors hope you'll join us and take the opportunity to join the J-Net community; it can support you in your learning process and give you the satisfaction of contributing to other engineers as well.

J-Net: Getting Started

1. Create a free account using your *personal email* so J-Net can follow you from job to job

2. Join the free Open Learning webinars that relate to your use of Junos

https://learningportal.juniper.net/juniper/user_activity_info.aspx?id=11478

3. Post questions about Open Learning sessions in the dedicated forum:

https://forums.juniper.net/t5/Training-Certification-and/Juniper-Open-Learning/td-p/313054

4. Post specific Junos operational or configuration questions under the appropriate technical category:

https://forums.juniper.net/t5/Discussion-Forums/ct-p/Forums

5. Utilize the free online vLabs to try out your new Junos skills:

https://jlabs.juniper.net/home/

6. Help others by reviewing and engaging with posts.

Thanks, and enjoy the book!

Steve Puluka, Network Architect @ DQE Communications, Juniper Ambassador

Introduction

When starting careers in network engineering, a large majority of network engineers typically start with devices manufactured by Cisco. There's nothing wrong with this as Cisco is a large networking vendor who makes quality network devices. It is important to remember, however, that they are not the only vendor of networking devices. There are many other competitors out there, some sell products for the small office/home office, some concentrate on enterprise networks, and others on the data center markets. One such vendor is Juniper Networks.

Juniper Networks has actually been around since 1996 and their goal was to take all the best parts of the networking world and make it better, combining the best possible throughput and lowest latency possible with ease of administration and ways to assist network engineers from recovering from configuration errors. From this was born Junos OS, more commonly known as Junos.

You may find it surprising to know that devices running Junos can be found in many different environments, from data centers, enterprise networks, Service Provider networks, small offices and probably the most significant network, the Internet. Devices from Juniper Networks can be found in large Internet Service Provider networks. In fact, if you downloaded an electronic copy of this book, the packets carrying the data probably passed through several Junos based devices.

It goes without saying that both the devices supplied by Juniper Networks and Junos itself are very different from other network operating systems, and that being the case, network engineers new to the Junos OS need to know how to install,

configure and support these powerful, flexible network devices. What network engineers need is a book they can sit down and read through. What network engineers need is this book.

This book was written specifically for network engineers who already know what ARP does, and the difference between TCP and UDP, but who need to know the basics of network devices from Juniper Networks and the basics of Junos itself. This book doesn't just stop at the basics, however, as it carries on to some slightly more advanced technologies such as routing the internet protocol, BGP.

Whilst network security is important, the primary focus of this book is the routing and switching area of technology, which therefore makes this book an absolute must read for those wishing to enter the world of networking with Juniper Networks, just as we, the authors, decided to do so all those years ago. Enjoy.

Sincerely,

Martin Brown and the Juniper Ambassadors

August, 2020

Chapter 1

Junos Fundamentals

by Martin Brown

The goal of network vendors is to move packets and frames around the network in the quickest and most efficient way possible, but each vendor achieves this goal in slightly different ways. Some devices use custom chipsets; others use generic off-the-shelf hardware, compiled in unique ways; some vendors create their own operating system; others rely on open source software (and perhaps add value by tweaking it in innovative ways.)

This individual approach is a good thing, as it allows for innovation and competition and allows customers to benefit from ever-increasing levels of technology that deliver faster and more reliable networks. On the downside, while a network engineer may know a particular vendor exceptionally well, if they are introduced to another vendor they may struggle at first – even if they are experts on how a packet is routed, a frame is switched, and how devices determine the best path across a network.

This book therefore begins by looking at Juniper's take on the network – or as Juniper calls it the *New Network*, as Juniper wants to redefine what a network can do with greater speeds and the latest technology. We'll start with a look at the different firewalls, routers, and switches that make up this new network before moving on to how the network operating system, which runs on these devices, is structured, and how the hardware is designed to maximize throughput, even when routing protocols are re-converging.

Welcome to the Juniper Networks Family of Products

It should be no surprise to hear that a vendor the size of Juniper has quite an array of different models and types of network devices fulfilling specific needs within the network. Many of these devices can be grouped together and classed as either a firewall, a router, or a switch; however, things are not always that simple. For example, some switches can also act as routers. Some firewalls can also act as switches. Even within those categories, there may be devices that are best-suited at the access layer, the core, or the border edge of the network.

Juniper gives each of their network devices a model number that begins with (up to) three letters that identify which family that device belongs to. These letters are followed by a number indicating the specific model of that device. As a general rule, the bigger the number, the more powerful the device. The family and model are typically printed on the front of the device. As an example, Figure 1.1 shows the front of an SRX320. SRX is Juniper's name for a huge suite of firewalls, from small office boxes all the way up to massive data center devices that are the size of vending machines.

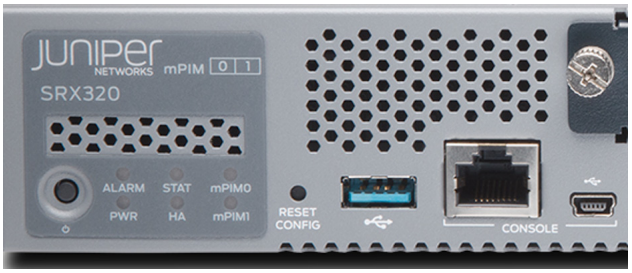


Figure 1.1 *Juniper SRX320 Firewall*

Let's take a moment to familiarize ourselves with the current family of devices and discuss where you'd expect each device to be used, starting with Juniper's EX Series range of switches, aimed at the enterprise market.

EX Series Ethernet Switches

A typical enterprise network, more often known as a "Campus LAN", is divided into three layers: the access layer, the purpose of which is to give network connectivity to devices such as workstations, printers and phones, the network core, which provides fast routing capabilities and connectivity to data centers and the internet.

Finally, there is the aggregation layer, which connects the access layer to the core layer. By dividing the network into three layers, traffic is allowed to flow in a logical and scalable way:

The official term for splitting the network up into these layers is known as the *Hierarchical Internetworking Model* and we can see an example of this in Figure 1.2.

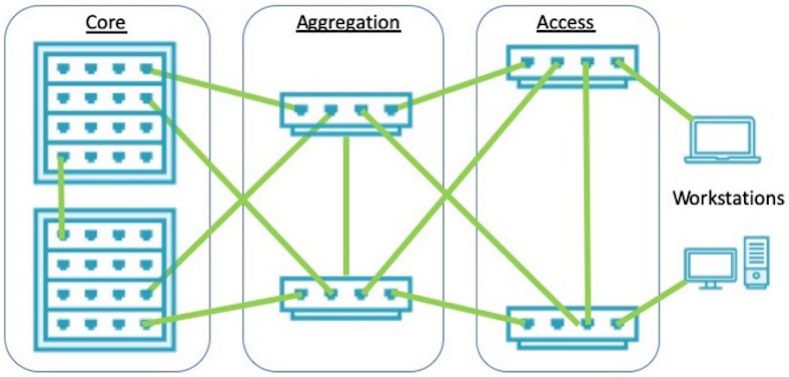


Figure 1.2 *Hierarchical Internetworking Model*

Each network layer moves packets and frames around the network in a slightly different way. For example, the access layer tends to be Layer 2, which means it sends traffic between devices within the same subnet or VLAN. In addition, it optionally marks frames with what is known as a class-of-service (CoS) marking – information used to tell the network whether this traffic is important and how important. As an example, CoS can be used to distinguish between voice traffic and data traffic.

By contrast, the aggregation layer routes packets between VLANs – in other words, it operates at Layer 3, where IP addresses are inspected. In addition, it also looks at the CoS marking that was set in the access layer and is able to prioritize packets during times of congestion on the network.

The core layer wants to do one thing: throw packets round the network as quickly as possible, and the aggregation layer helps the core layer do just that. We won't be doing anything intensive like re-marking the CoS settings on the packet, or checking firewall filters to see if the traffic is actually allowed. The core network is built for speed: packets must leave this part of the network as quickly as possible. Deep inspection always happens further down in the layers.

With that in mind, the EX Series of switches was designed specifically for enterprise networks. Some models were designed to sit at a specific part of the network, for example as access switches. Other models are powerful or flexible enough to have a dual purpose and can be used as access or aggregation switches depending on how

large your network is and how much traffic will flow through the device. All switches can be used to bring high-speed LAN connectivity to the enterprise network. In addition, some models can also act as basic routers, potentially acting as an exit point to the internet, or to other data centers.

The models start at the EX2300, which is available in 24 or 48 port models and is capable of a throughput of up to 178-Gbps. If you'd like a small fan-less version then the 12 port EX-2300-C, or compact, is also available. These models were all designed for small offices and the access layer.

In case you were wondering, device throughput is a theoretical calculation based on the number of ports in the device and the maximum speed of each port. Because each port is bi-directional, this figure is doubled. As the EX2300 can have up to 4 x 10Gbps ports, 48 x 1Gbps ports, and 1 x 1Gbps management port, when added together and doubled gives you the maximum throughput of 178Gbps. This means all users could send massive amounts of data, but all users should still get a positive experience that won't be affected by the traffic of other users.

Other switches that were designed to be used in the access layer include the EX3400 and EX4300. In the aggregation layer one would typically find EX4600 and EX4650 switches, although should you wish to, you can also use these switches in the access layer.

Finally, the EX9200 and EX9250 are fast, powerful switches that were designed specifically for use in the network core and are capable of a throughput of up to 4.8Tbps.

Most switches support both copper and fiber connections. The EX4300, for example, has fixed copper ports, but it also has an optional plugin module, which supports fiber connections that can be added either during the initial install or at a later date. This module can be seen on the far right of the EX4300 switch in Figure 1.3.

Figure 1.3 Juniper EX4300 48 Port Switch

QFX Series Switches

While most of us have a pretty good postal network where a letter or parcel could be sent anywhere in the world in a week or even a matter of days, imagine for a moment that the postal service in your country started to offer a service where a letter was collected and delivered by supersonic jet – albeit for a slight increase in price. That same letter could be delivered anywhere within a matter of hours.

Now imagine that the downside to this service was that the letter or parcel wouldn't be collected from you for several days. The time taken in collecting the letter would negate the advantage gained by the extremely fast transit time. In networking terms, this delay is known as latency, and instead of being measured in days, latency is measured in milliseconds.

Switches over the past few years have seen a huge increase in speed with greater throughput as interface speeds have gone from Gigabit to 10 Gigabit to 40 Gigabit, with 100 Gigabit becoming the norm in service provider cores. However, this speed advantage is lost if there is too much latency. This is where QFX switches come in.

Where the EX Series was designed for the enterprise network, QFX Series Switches are designed specifically for high speed data centers. At first glance some QFX switches could be mistaken for an EX switch but looks can be deceiving. QFX switches offer ultra-low, often sub-millisecond latency in addition to high speed connectivity between nodes using a protocol developed by Juniper called QFabric.

Figure 1.4 shows an example of a QFX10008 switch. This particular switch is comprised of 36 QSFP ports, with each port being capable of speeds of up to 40Gbps. SFP stands for *Small Form-Factor Pluggable*, and the Q stands for *Quad*. The idea is that these ports can either run as 40Gbps ports or be split into four 10Gbps ports using something called a breakout cable. This gives you great flexibility to decide whether to place it in the core of your data center, or whether instead to place it somewhere like the access layer in the enterprise – except that this time, this layer will be home to many servers. In data centers, the access and core layers are often referred to as *leaf* and *spine*.

Figure 1.4 Juniper QFX10008 Switch

MX Series Routers

Chances are you have an VSDL router or cable modem at home which allows you to access the internet. It's probably small, cheap, plastic, and does the job perfectly well.

However, if that same router were to be placed into an enterprise network, it would struggle. At the very least, your home router probably can't handle 100 devices on the network at once, whether in terms of the throughput or even just in terms of being to remember that many devices all at once. In addition, our enterprise router is probably offering a lot more functionality, such as learning where multiple IP networks are, or offering basic filtering functionality. Immediately you can see the need for enterprise networks to have much more powerful routers than the average household.

Similarly, if we compare the needs of that enterprise with the needs of an ISP, we would once again find that the ISP routers have vastly different requirements. ISPs will probably be less interested in filtering traffic – that will be left to dedicated devices elsewhere in the network. Instead they'll be much more interested in learning huge numbers of different routes, or prefixes as we sometimes call them (because the first section of the IP is often the bit we're interested in). These devices will of course cost significantly more than devices aimed at the enterprise.

With all that in mind, let's talk about Juniper's MX Series of routers, which range from the tiny MX5, perfect for enterprises and the edge of an ISP network, all the way up to the MX2020, a device so big that it's probably taller than most people! Some Ambassadors even have MX Series routers in their home lab.

The modest MX5 offers 20Gbps maximum throughput. By contrast, the MX2020 is capable of up to 80Tbps. At the time of writing, there are sixteen different MX series model, including the vMX, which is used for routing packets in virtual environments such as private and public clouds, and can also be used in labs to get hands-on experience with an MX router without having to buy the hardware.

Figure 1.5 shows an example of an MX240 router. This router is in a format known as a *chassis*. This format means the router is comprised of blades that can be added or removed, sometimes even without powering down the chassis itself. You can buy different MPCs that offer different functionality based on your specific requirements. For example, an MPC is available that contains just switch ports, while other MPCs are designed for connectivity to fast fiber or serial links. Other MPCs bring extra functionality to the chassis beyond what it's normally capable of, such as enhanced encryption and security capabilities.

Figure 1.5 *Juniper MX240 Router*

ACX Series

While MX routers are a universal routing platform, Juniper also manufactures several routers that are more bespoke, tailored to a particular need or purpose. There are three such routers. The first is the ACX Series router, also known as a *universal metro router*.

In an enterprise campus LAN, most of the internal traffic will be sent as an Ethernet frame over copper Cat 6 or 7 cable, or over fiber if the frame is sent between switches or to a server. This type of network is relatively straightforward.

A network for a Telcos, or telecommunications company, is much more complicated than the network of an enterprise LAN. While they are similar in that the core of the telco network has to be fast, the difference is how spread out the telco network is, and more importantly, the environment in which network equipment is installed, which sometimes can be quite harsh.

For example, consider your cellular phone or a smartphone. When you wish to make a call, send a message, or just send data, your phone will connect to a cell tower in your area. Once the signal has reached the cell tower, the data must then somehow be sent to the telco's network and the link that connects the cell tower to the telco's network is known as a *backhaul* (see Figure 1.6).

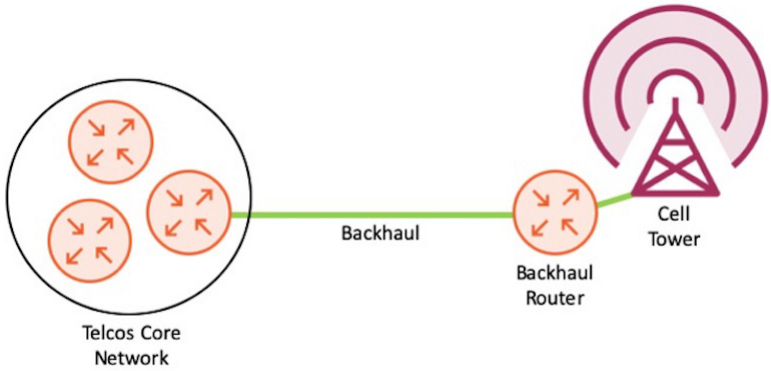


Figure 1.6 *Backhaul Example*

One important feature about cell towers is that they tend to be located outdoors, so imagine that when it rains or snows, or on really hot days in the summer, that cell tower takes the full force of nature, and to some extent, so does the network equipment located at the bottom of the cell tower.

ACX routers were designed with the backhaul in mind, and while the ACX500 supports a throughput of up to 60Gbps, the router itself is smaller than a typical router; it's hardened, and as it doesn't have a fan, and there are no moving parts. This makes the router less likely to fail even in the harshest of conditions. It's even possible to deploy an ACX500 outside, as the ACX500-O was designed specifically for outdoor installations.

While some of the routers and switches we've looked at so far are available in a chassis form, which can be quite large, the largest ACX, which is the ACX4000, is only 2.5 U in height. It's also worth noting that most ACX routers have been environmentally hardened and have fanless passive cooling. The only exception to this is the ACX4000.

NOTE Devices are often measured in units of U – a standard number that says how tall the box is, so that you can correctly identify whether your rack will have room for the device.

CTP Series

A number of years ago, before VDSL or even ADSL had been developed, organizations needed an alternative way of connecting to their branch offices. Dialup modems were an option but they were too slow and too expensive to be connected all the time. ISDN was faster, but again, too expensive to be always on. The best alternatives were serial links, which supported up to 8Mpbs, or T1 lines that supported speeds of 1.544Mbps. In Europe, E1 lines were used instead of T1 lines and E1 lines were faster, supporting speeds of 2.048Mbps.

Networks have progressed considerably since those days. Most organizations will at least have a VSDL line, a fiber connection coming into their premises, and have their data carried on Synchronous Optical Networking, or SONET, fiber optic networks that support speeds from 51.84Mpbs up to 200Gbps.

On that basis, you'd be right to think that serial links, E1, and T1 lines were a thing of the past and as such are no longer in use, however, there are a few legacy applications that require their use and because of that very reason, the CTP Series of routers was developed, an example of which is seen in Figure 1.7.

Figure 1.7 *Juniper CTP151 Router*

On the right side of the CTP router are two plug-in modules. The module furthest to the right is for connections to serial links and the module on the left is for connection to E1 and T1 lines.

The CTP series wasn't just developed for connections to legacy networks. Typically, when data is received on a serial link, a router would strip any encapsulation from the frame and send the data as a packet inside an Ethernet frame. CTP routers don't do this, instead they leave the frame intact and send it exactly as the data was received but inside an IP packet.

Each router is capable of connecting to multiple serial links or E1/T1 lines and as such, the router can encapsulate multiple streams from multiple serial ports inside the same IP packet. This process is known as *time-division multiplexing* (TDM) and is used to make sending the data over the IP network a more efficient process.

The CTP Series allows a service provider or telecommunications company to replace the old legacy infrastructure that's used just to support serial links or E1/T1 lines with a modern IP network, while still offering the legacy services to customers that require it.

PTX Series

Almost all devices on a network, whether they be in the enterprise, in branch offices, or in the home, will send data as a Layer 2 frame to a switch. The switch then acts as a bridge. It will inspect the destination MAC address, and after consulting its MAC address table it will either forward the frame directly to another host, or to a router, if the sending device has determined that the destination is on another subnet. Alternatively, the network will utilize a wireless network in conjunction with a switched network.

While service provider networks are similar to enterprise networks in that they still carry frames and packets, the main difference between service provider networks and enterprise networks is that the enterprise LAN will typically carry the network traffic for a single organization, whereas service providers have traffic from multiple organizations traversing their network. Organizations need a way of connecting their HQ to their regional offices and off-site data centers, and service providers are used to achieve this.

The issue service providers faced was how to keep customer traffic separated, preventing one customer snooping on another customer's data, and how to prevent a subnet used by one customer from conflicting with another customer without having to tell each customer to change their address space. The solution was to use *Multiprotocol Label Switching*, more commonly known as MPLS.

This Beginner's Guide to Junos doesn't require you to know about MPLS, but let's take a moment to talk about it anyway, because not only is it great to know what MPLS is, it will help you to understand the value of Juniper's PTX Series.

MPLS adds an additional header between the Layer 2 and Layer 3 headers (why MPLS is sometimes referred to as layer 2.5). This is known as a *shim* header and this is added to frames that have been received from a customer after the data has been processed by the service provider's edge router but before the data is sent into the service provider's core network.

The customer router that connects to the service provider's network is known as the *customer edge* or CE router, and the service provider's router that connects to the customer's network is known as the *provider edge* or PE router.

When the PE router receives the frame, the PE router removes the Layer 2 header, looks at the destination address in the Layer 3 header, refers to its routing table and determines where the data needs to go. Before forwarding the packet into the service provider's network, the PE router attaches, or pushes, a unique label into the shim header.

Routers inside the service provider network will always refer to the label in the shim header and not the IP address in the Layer 3 header. By using a label for routing data, customers sending data over the service provider's network can use the same IP address space without conflicting with one another. In addition, because the PE router will only forward data to a PE router connected to the same customer, there is no possibility that one customer can accidentally receive data destined for another customer.

The routers that receive the frame with the shim header already attached are known as P routers or even LSR's. These will examine the header and then swap the label before passing onto the next P or PE router. An example of how CE, PE, and P routers are connected is shown in Figure 1.8.

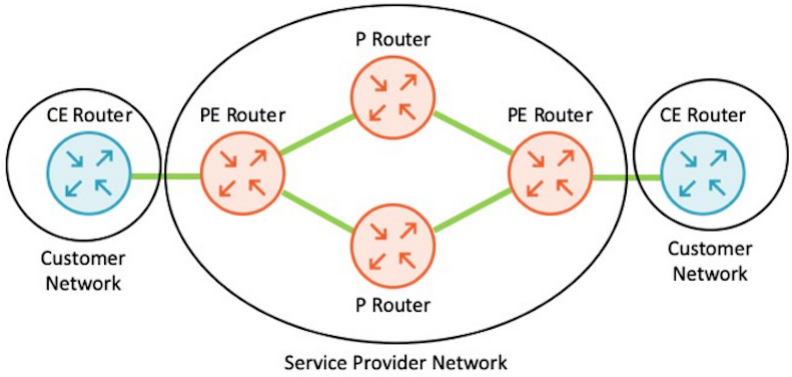


Figure 1.8 *Service Provider Network Example*

NOTE As you can imagine, MPLS is quite a complex topic and we've only just brushed the surface in this chapter. Should you wish to learn more about MPLS, then we strongly recommend visiting the TechLibrary or exploring other books.

While what's been said so far in this section may seem a little off topic, there is a good reason for it: the final bespoke router available from Juniper is designed to be placed in a service provider's core network. These routers were specifically designed to be P routers.

Known as PTX Series routers, they were created to optimize the core of a service providers MPLS network, tuned to perform the label lookup and label swap process better than the equivalent MX series routers but without an increase in power consumption.

PTX routers start with the PTX1000. This router has the option of up to 288 10 Gbps ports, 72 40Gpbs, or 24 100Gbps ports in a relatively small 2U chassis and has a throughput of up to 2.88Tbps.

At the higher end of the series is the PTX10016. This router is nine times taller than the PTX1000, taking up 18U in a rack. The router is supplied as an empty chassis so that the service provider can choose up to 16 individual line cards to create the router they want. These line cards support interfaces of up to 100Gbps and 400Gbps and each line card is capable of up to 14.4Tbps. This means the router has a maximum theoretical throughput of up to 230.4Tbps.

Figure 1.9 is the PTX10008, which is the same router as the PTX10016, except it only has slots for eight line cards instead of sixteen. This means it's half the height, with half the throughput, of the PTX10016. You might notice that in addition to the eight line cards, there are two additional slots at the top of the router. These are taken up with what are known as *Routing Engines* (REs), which are used to manage and control the router. These slots cannot be used to route traffic as the router will not function without its RE.

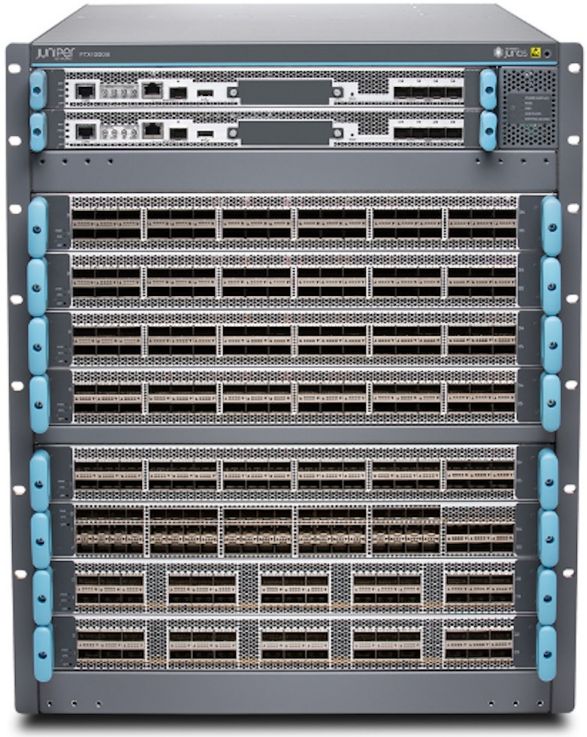


Figure 1.9 *Juniper PTX 10008 Packet Transport Router*

We'll discuss the RE in a bit more detail later on in this chapter, but let's move on to Juniper's vast firewall offering.

SRX Series

So far in our journey of discovery, the appliances we've discussed have either been routers or switches. There is one type of appliance, however, that is arguably as important as routers and switches, and that's the security appliance.

A number of years ago some bright spark decided that it would be a good idea to connect all networks together into some sort of interconnected network, and as such invented what we now know as the internet. Not long after this eureka moment, someone equally as intelligent realized that while data could be shared across the globe more easily, it also meant that someone with bad intentions could connect to any network and help themselves to secret information. From this, the firewall was born.

Firewalls have two main purposes: to stop people who shouldn't be accessing your network from accessing your network, and to allow people inside the network to access resources that are outside the network, such as a website.

Early attempts at this security had an inherent problem; they had no awareness of the sessions passing through them. To see why this is a problem, consider a security device that allows all traffic out from the LAN to the internet, but doesn't allow anyone from the internet to access the LAN. Seems like a reasonable setup – until you realize that it means you can't visit your favorite websites. Your request will certainly go to the website – but the reply from the website will be blocked. After all, it's coming from the internet!

These types of firewalls were known as *stateless* firewalls, due to the fact that they weren't able to keep track of the state of the session. Such firewalls still exist today: they're exactly the kind of so-called firewall rules that you can enable on a router or a switch.

By contrast, actual hardware firewalls are known as *stateful* devices and they are flow-based not packet based. They're intelligent and powerful enough to realize that if you enable traffic from your LAN to the internet, then of course you want the internet to be able to reply. It's only the brand new sessions coming from the internet that you want to reject.

Another shortcoming of these earlier firewalls is that, while they were great at keeping people out, when it came to restricting certain types of traffic from leaving the network, firewalls could only classify traffic by IP address, protocol, and port. They were unable to prevent network users from accessing inappropriate websites. In order to correct this, *proxy* servers were developed so that website requests went to the proxy server first and the proxy server would decide if the user was allowed to access that website.

Pretty soon someone realized that the internet could also be used to allow people who were on the road, staying in hotels, or even working from home to access internal corporate resources and as such *remote access virtual private networks*, or VPNs, were developed in order to allow this whereas previously modems were used to dial into a network via a phone line. VPN concentrators were used as a termination point for VPNs.

Thankfully, technology has moved on a lot since the early days of networks, and modern firewalls are now capable of doing more than just filtering traffic. For example, they are all stateful, and they can be used as a VPN endpoint for both remote access and site-to-site VPNs. Or they can prevent access to inappropriate websites by performing URL filtering, thus replacing proxy servers. These newer firewalls are called next generation firewalls, or NGFW, for short.

The SRX Series of firewalls are classed as NGFWs, and are also capable of supporting many other security features such as anti-virus and anti-spam. They are also able to detect attackers attempting to use vulnerabilities in software to gain access to a network. This detection process is known as an *intrusion prevention system*, or IPS, and Juniper's brand name for their offering is called *advanced threat protection*. ATP is capable of even detecting unknown viruses and security threats, thanks to its ability to receive real-time threat updates from the cloud and compare new attacks against an enormous database of known attacks. This suite of security features is known as *unified threat management*, otherwise known as UTM.

In addition to being a security appliance, the SRX Series are also very capable routers, able as they are to support different routing protocols, performing prefix filtering, and route redistribution. There have been several occasions where network engineers have installed an SRX appliance with the firewall disabled into an enterprise network, in effect using the SRX just as a powerful router, such is the attractive price point of the entry level SRX. A world class firewall, or a world class router: two options in one box!

At the time of this writing, the current range of SRX Series firewalls start with the SRX300. These firewalls are designed for small enterprise networks and branch offices. They are capable of processing VPN traffic at a rate of 1Gbps and are able to inspect traffic for security violations at a rate of 2Gbps.

At the other end of the scale is the SRX5800. This firewall is designed for high-end data centers and service providers and is capable of routing packets at a rate of 2Tbps. If this firewall were to be used as a VPN endpoint, then it could process VPN traffic at a rate of 200Gpbs. The SRX5800 supports the same security features as the SRX300, however, it is capable of inspecting traffic for any security breach attempts at a rate of 100Gbps.

Juniper also offers virtual SRX appliances for cloud applications. The vSRX and the cSRX. The vSRX is designed for use in public clouds reachable via the internet, such as Amazon Web Services or Google Cloud, and private clouds such as VMWare ESXi, which normally run on a private network. The vSRX is used to provide security for virtual appliances.

The cSRX on the other hand is known as a *container* firewall. Containers are part of an automation system called Contrail. The cSRX can be spun up or spun down on demand in less than a second. One major difference between the cSRX and the rest or the SRX series, is that the cSRX only filters traffic between Layers 4 and 7 of the OSI model while the rest of the SRX series can filter traffic at Layers 2 to 7.

It's helpful to note that the vSRX is very useful in a lab environment for those who want hands-on experience with an SRX, but don't want to buy new hardware just for the privilege. The cSRX, on the other hand, would only be useful to engineers who are experimenting with automation as there is very little in the way of a user interface.

Software Architecture

The Junos OS is what is known as a network operating system, but in reality it is more like an application that runs on top of another operating system, rather like how Putty runs on Microsoft Windows. Junos, however, obviously doesn't run on Windows.

When Juniper was designing Junos they had a requirement for an underlying operating system that was stable, easy to secure, and one that didn't require too much in the way of system resources just to run the operating system itself. Juniper settled on an operating system known as FreeBSD, which is an open source OS based on UNIX.

One unique feature of UNIX-based operating systems is the way they run *daemons*, which are very much like services. These run in their own memory space, independent of other daemons. Running daemons in a separate memory space helps protect Junos in the event a daemon fails. Should a daemon go rogue and crash, it is very unlikely it will cause the entire operating system to crash. What this means from a Junos perspective is if a daemon crashes in a catastrophic manner, only that daemon is affected. A network engineer could easily restart the affected daemon without having to reboot the entire network device.

Another benefit of having Junos run on a UNIX based operating system (as opposed to Junos itself being the underlying OS) is that the underlying OS can also be used for maintenance purposes, such as backing up files, viewing logs files, and copying new Junos updates to the flash memory on the network device prior to an upgrade.

More recently, Juniper has started to use Linux as the underlying operating system. This operating system offers the same level of stability and security as FreeBSD, however, Linux is more flexible as it lends itself more open to features such as automation.

Every device mentioned in this chapter runs Junos, regardless of whether it's a switch, router, or firewall. This gives network devices manufactured by Juniper a unique advantage: if a network engineer learns to speak Junos using a switch, they will have no issues configuring a Juniper router, or firewall. Juniper refers to this as *One Junos*.

What One Junos means is that most of the commands, whether they are used to configure the device or used to detail the status of a network device, are the same across all types of Juniper network appliances. There are a few exceptions to this rule, such as where a network device doesn't support a particular feature: for example, where a low-end EX series switch might not support MPLS.

Aside from these rare exceptions, the look and feel of Junos is common across all platforms. This is beneficial because whenever an organization receives a new Juniper network device they've never used before, the network engineer who will be installing, configuring and supporting the new device will not require any additional training.

Control, Forwarding, and Management Planes

Imagine a child running up to a mom shouting Mom, I really want this… while at the same time another child is also running up to the same mother shouting Mommy, I hurt my knee. For a moment the mother can't quite make out who is shouting what. Usually this situation is followed by the mom shouting at them to be quiet so she can gather her thoughts.

This scenario could, in theory, be described as a *denial of service* attack against a mom where her resources are overwhelmed with incoming requests. You are probably thinking what's this got to do with networking? That's a reasonable question to ask.

So, let's imagine for a moment that the mom in our scenario is a router and the children are packets of data. One packet of data is carrying a phone call and the other packet is carrying an update from a routing protocol. What would be less than ideal is for the router to have to stop the phone call briefly just because someone has added a new subnet somewhere on the network.

One solution to this could be to have a fast multicore processor such as an Intel Xeon E3, which can process data much faster. There are drawbacks with this, however, such as increased cost, increased power usage, and heat. In addition,

while Xeon E3 may be great in a server, it's not really optimized for making decisions as to what port to send out packets of data to.

Another more practical solution is to identify the purpose of the packet, then split the router into multiple parts and have one part, say, handle the packets from the phone call and have another part deal with routing protocol updates. To return to our metaphor, there's a need for more than one mom.

Routers, EX and QFX switches, and SRX firewalls are internally split into multiple parts known as *planes* and there are three planes in each network device: the *control*, *management*, and *forwarding* planes and they are joined together by very fast internal links.

The forwarding plane, also known as the *data plane,* has the responsibility of handling data that's just passing through the device, for example, where data is being sent from one node on the network to another separate node somewhere else on the network. When a router or switch receives a packet or frame, the forwarding plane will examine the destination address in the header of the packet or frame and immediately determine which interface the data needs to be sent out from.

In order to achieve this in the fastest and most efficient way possible, Juniper designed a special processer known as an *application-specific integrated circuit* (ASIC). This processor contains a list of all destinations known to the device. When a packet or frame enters the device, the ASIC examines at the destination address, consults its list, checks which interface the destination is reachable via, and throws it out of that interface all in less than a millisecond.

ASICs are the perfect processers for forwarding packets of data at very high speed. In addition, they are very efficient and use a minimum amount of power, but it's also worth knowing that this type of processor would be absolutely useless at running Microsoft Windows.

Control planes, however, have a very different task to do. They receive traffic sent from other network devices where the destination is the router itself. This traffic is typically from a routing protocol advertising a new route, or just reminding the router that a route is still reachable.

When a router receives a routing update, it usually has to run some sort of algorithm to determine the best path to that new subnet. This can require some heavy processing and as such requires a different type of processor capable of performing extensive mathematical calculations. Some control plane processors were actually x86 based, and indeed the control plane on the older J Series routers were powered by an Intel Celeron processor; this meant the control plane could in theory run Microsoft Windows, albeit without a graphics card.

The final plane is the management plane, and as the name suggests this allows the device to be managed by something, not necessarily someone. As with the control plane, the router is watching for traffic destined for itself. If the router determines

this traffic is not a routing update but is a management protocol of some sort, such as SSH, the router will pass the traffic to the management plane, the management plane will then either deny or allow the traffic, and if the traffic is allowed it will process the traffic accordingly.

As mentioned, management traffic might not be someone connecting to the device to make changes but could be a monitoring server, such as a *simple network management server* (SNMP), checking that the device is operating normally, that it's not running too hot, or simply checking it's still alive. In addition to this, some devices also send ICMP requests, or pings, to make sure their neighbor is up and if there is no response, the router could withdraw a static route from the routing table. ICMP requests could be handled by the management plane.

The reason for this separation is simple, as with a mother being overwhelmed with requests from her children, a router can quickly become overwhelmed if a routing protocol would suddenly send a huge database that needed processing. In addition, an attacker could quite easily bring your network to a halt by sending lots of spoof management traffic to the device.

By separating the multiple planes, the router would be able to continue forwarding data to users and other nodes uninterrupted, even if an attacker was attempting a denial-of-service (DoS) attack on your router or if the router had suddenly received a huge BGP update which needed processing.

Transit Traffic and Exception Traffic

Traffic that is processed by a router or a switch is divided into two distinct categories; the first category is *transit* and the second category is *exception*.

Transit traffic is pretty much any traffic where the source and destination are anything other than the router the traffic is transiting though. For example, let's say a laptop with an IP address of 10.0.10.10 sent a packet to a server with an address of 192.168.1.23, as shown in Figure 1.10. The routers in the path are neither the client nor the server and instead they are just being used to forward the traffic through the network. To these routers the packet being sent from the laptop is transit traffic.

Figure 1.10 *Transit Traffic Transiting Routers*

Referring to the previous section for a moment, the plane that processes transit traffic is the forwarding plane. Transit traffic isn't just limited to a conversation between devices, also known as *unicast* traffic. If traffic is being sent from one device to many but not all, it is known as multicast traffic. Multicast traffic is typically transit traffic too, unless the traffic contains a routing protocol update.

On the other hand, *exception* traffic is, from the perspective of a router or switch, classed as traffic where the destination address is either that of the router or switch itself (where the traffic has originated from the router or switch, or any other traffic that the forwarding plane is unable to process). This traffic is handled by either the control plane or the management plane, depending on what kind of traffic it is.

The reason why this type of traffic is referred to as exception traffic is because it doesn't follow the normal traffic flow rules. When a router receives exception traffic, the router doesn't need to perform a route lookup, it doesn't need to determine which interface the packet needs to be forwarded out of, and traffic will be unaffected by firewall filters attached to an interface; this traffic is an exception to the rules used for transit traffic hence the name exception traffic.

As an example, imagine a user is connecting to router with the name ACME-HQ-RTR-01 via a web browser as per Figure 1.11. From the perspective of router ACME-HQ-RTR-01, this web browser traffic destined for itself is exception traffic.

Figure 1.11 *Exception Traffic Entering a Router*

There are several types of exception traffic, and as mentioned before this traffic can either be processed by the control or management planes. Examples of traffic directed at the router to be processed by the control plane include:

- Processing updates from routing protocols
- Sending routing updates to routing protocols

Routing protocols supported by Junos include RIP, OSPF, IS-IS, and BGP. A summary of each of these routing protocols is provided later in this book.

However exception traffic, which will ultimately be processed by the management plane, includes:

- SSH, Telnet, HTTPS, and HTTP used to configure the device
- SNMP GETS

- VRRP hellos and keep-alives
- NTP requests both to and from the device

NOTE Don't worry if you don't know what some of these traffic types mean, by the end of this book you should have a much better understanding of most of the above traffic types in the list.

As an interesting twist, it is also useful to remember that it's perfectly feasible for some traffic to be classed as transit traffic as it traverses one router, only to become exception traffic when the traffic reaches another router. For example, imagine a laptop with an address of 10.0.10.10 is connecting to the router ACME-GLA-RTR-01via Telnet, but in order to reach this router the traffic must pass though several subnets first. An example of this is shown in Figure 1.12, where routers A and B would class the traffic as transit, whereas ACME-GLA-RTR-01 would class the traffic as exception.

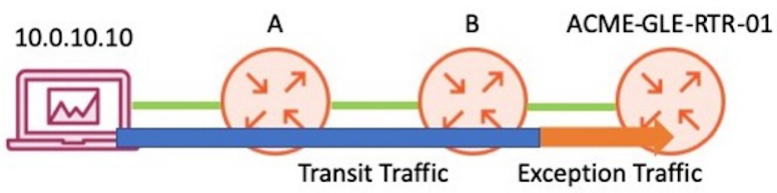


Figure 1.12 Transit Traffic Becoming Exception Traffic

Routing Engine and Packet Forwarding Engine

As mentioned earlier in this chapter, network devices are divided into planes, forwarding for data, control for routing protocols, and management for configuring and monitoring the device; you would be forgiven if you thought that this was a logical division, however, the division is actually in hardware, and for a very good reason.

All network devices running Junos, whether it's a small EX2300-C or a powerful PTX10008, are physically divided into two internal components: the RE and the *Packet Forwarding Engine*, or PFE.

The RE is pretty much the brains of the device. Its main purpose is to calculate the best network path to each individual subnet based on input from various routing protocols and static routes. Sometimes this calculation is quick and easy if the network device is running in a small branch office, but if the network device is routing part of the internet or is located in a service provider's core, these calculations can be complex. The RE was designed for fast number crunching.

The control and management planes are part of the RE, therefore if a network engineer connects to the device and starts making configuration changes, the engineer is actually configuring the routing engine. When a routing protocol sends a network update to the network device, the network update will be received and processed by the RE.

In addition to taking care of the control and management planes, the routing engine also takes care of the most important part of a Juniper network device: Junos itself. Basically, Junos is part of, and is running in, the routing engine.

As mentioned, there are two components, the first is the RE and the second component is the PFE. As its name suggests, the PFE controls the forwarding plane and its task is to either route or switch packets or frames out of the correct interface as quickly as possible, which is one of the strengths of Junos-based devices; they have a very high throughput compared to some of their competitors.

As mentioned, when we discussed the forwarding plane earlier in this chapter, *application-specific integrated circuits* (ASICs) are the power behind the PFE. These programmable processors are designed specifically for routing and switching packets and frames. However, the PFE isn't particularly bright; it's very fast and very strong, but it needs to be told what to do as without being programmed, it will just discard any packets or frames it receives. This is where the RE comes in, the RE will program the ASIC so that the PFE can go about doing what it does best.

The process for telling the PFE how to process packets is as follows:

1. One of the following will happen:

    a. The routing engine will learn new routes from a routing protocol.

    b. A network engineer will make a change to the device.

2. The RE will process this information, perform several calculations, and create what is known as a *routing table*.

3. The routing table is then processed to include exit interface information and is then copied to what is known as a *forwarding table*.

4. Once the forwarding table has been populated, a copy is then sent to a forwarding table that exists on the PFE.

This process is demonstrated in Figure 1.13

NOTE In case you aren't sure of the purpose of a routing table and forwarding table, don't panic, these will be discussed in detail in Chapter 5.

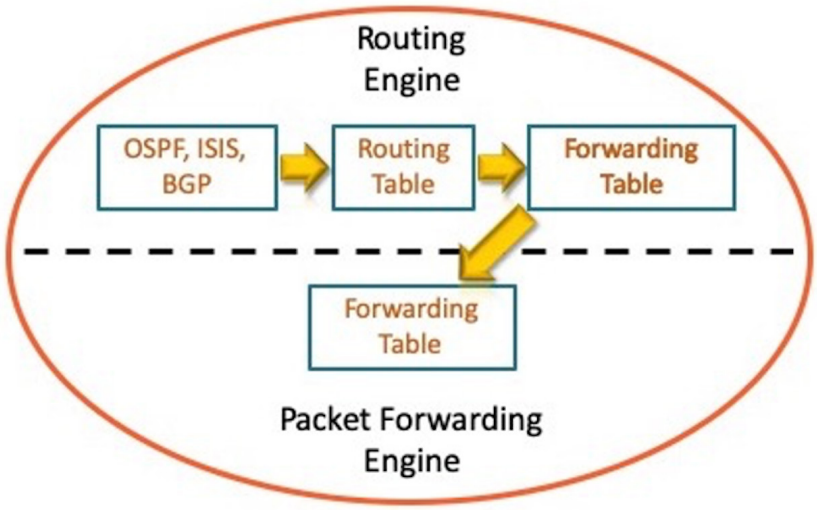


Figure 1.13 RE to PFE Programming Flow

In order to facilitate this process an internal link joins the RE and PFE. It is very fast and is bi-directional. If an interface receives a packet destined for the RE, for example, a message from the routing protocol, the PFE will process this packet first and it will then forward it to the RE via this internal link.

Typically, most of the traffic the network device receives will be transit traffic. In addition, ICMP echo requests and replies to the device are also handled by the PFE. Although if an ICMP is unreachable or a TTL expired needs to be sent, then the PFE will need to pass this to the RE and the RE will send the ICMP error messages instead.

The same can also be said for what are known as *IP options*. As you know, IP headers contain information such as source and destination IP addresses, and the type of protocol such as TCP or UDP. In addition, the header also has a field known as IP options. This field is used to tell the recipient of the packet and any routers in the path any pertinent information, for example IP option 18 identifies that the packet is being used for trace route. Usually the PFE is able to handle IP options, but occasionally a header will contain an obscure value and in that situation the packet is sent to the RE to be processed.

As mentioned, the PFE and RE are joined together via an internal link. Because this link is internal and directly connects these two parts, it is quite fast and sends data from the RE to the PFE very quickly, however, this speed has the potential to be its Achilles heel. If an attacker started sending hundreds of packets that contained obscure IP options or that made it necessary for the router to send ICMP

unreachable messages, the link could flood the RE with messages causing it to become overwhelmed. In other words, there is the potential for a DoS attack to be performed against Junos-based devices.

In order to prevent someone from performing a DoS attack against a network device running Junos, Juniper engineered rate limiting into the link between the RE and PFE. If the PFE sends too much traffic to the RE, some of the traffic will be dropped. There is no real need to rate-limit traffic being sent from the RE to PFE.

If the RE was down for some reason, say Junos was being upgraded with new software or there was a major issue, a major benefit of the physical separation between the control and forwarding planes is that the PFE could continue, unaffected, to forward transit data to other network devices and nodes. Once the RE returned to normal function, the PFE would then resume forwarding exception traffic to the RE.

It is actually quite rare for a RE to suffer a complete catastrophic failure, however, and the reason for this stability is partly due to the way the underlying operating system uses daemons.

Protocol Daemons

As mentioned earlier in this chapter, a daemon is very much like a service in Microsoft Windows. As they run in their own separate memory space independent of other daemons, this helps protect the entire operating system and other daemons in the event one of the running daemons suffers from a crash.

A network device that is running Junos will be running quite a few daemons. In the Junos CLI you can use the command `show system processes` to list the daemons that are currently running. In addition, if the option `summary` is included, Junos will report how many daemons have been started.

NOTE Although the correct term for the services that run in UNIX is daemons, Junos refers to a daemon as a *process*. This is reflected both in the `show` command and the output that is generated:

```
root@ACME-HQ-SRX-01> show system processes summary
last pid:  2241;  load averages:  0.15,  0.06,  0.01  up 0+01:42:40    13:31:40
134 processes: 14 running, 107 sleeping, 1 zombie, 12 waiting

Mem: 188M Active, 117M Inact, 1053M Wired, 211M Cache, 112M Buf, 402M Free
Swap:


  PID USERNAME  THR PRI NICE   SIZE    RES STATE  C   TIME   WCPU COMMAND
 1616 root        5 139    0  1026M 82244K CPU1   1 109:09 93.95% flowd_octeon_hm
   22 root        1 171   52     0K    16K RUN    0  81:54 82.67% idle: cpu0
```

From this output you can see that the device that this command was run on, an SRX-110, has a total of 134 daemons that have been started. But of these 134 daemons, only 14 are actually running and 107 are sleeping, which could mean a feature has been configured but isn't being used at this time, and as such FreeBSD and Junos puts the daemon to sleep.

If the command `show system processes` is entered without options, the entire list of daemons is shown in the output. This list can be quite extensive and in addition some of these daemons are associated with the underlying OS. In order to identify which daemons are part of Junos, you need to look for a path in front of the name of the daemon in the output. If the path starts `/usr/sbin/`, then this daemon is part of Junos:

```
root@ACME-HQ-SRX-01> show system processes
  PID  TT  STAT      TIME COMMAND
    0  ??  WLs    0:00.00 [swapper]
    2  ??  DL     0:02.91 [g_event]
    3  ??  DL     0:05.38 [g_up]
    4  ??  DL     0:05.04 [g_down]
    5  ??  DL     0:32.14 [rtfifo_kern_recv]
    6  ??  DL     0:00.08 [kqueue taskq]
    7  ??  DL     0:00.03 [thread taskq]
!
output omitted for brevity
!
 2093  ??  S      0:03.06 /usr/sbin/httpd --config /jail/var/etc/httpd.conf
 2216  ??  S      0:06.30 /usr/sbin/rpd -N
 2576  ??  S      0:00.97 /usr/sbin/rmopd -N
 2579  ??  S      0:01.06 /usr/sbin/smihelperd -N
 2690  ??  R      0:00.04 /bin/ps -ax
    1  u0- SLs    0:01.79 /junos/sbin/init -D/junos --
 1232  u0- S      0:00.08 /usr/sbin/usbd -N
 1247  u0- S      0:01.52 /usr/sbin/eventd -N -r -s -A
 1649  u0  Ss     0:00.13 login [pam] (login)
 1743  u0  S      0:00.15 -csh (csh)
 1757  u0  S+     0:03.02 cli
```

The output from the `show system processes` command can be filtered to display only the daemons that start `/usr/sbin/`. Later in this book we discuss how to filter output in order to display only relevant information.

While some of these daemons are related to the underlying operating system, quite a few are part of Junos, and as a network engineer, it is useful to become familiar with the most important daemons and their purpose. Table 1.1 lists the daemons you need to be aware of, their full name, and a description of their purpose.

Table 1.1 *Important Junos Daemons and Their Purpose*

Daemon	Name	Description
`rpd`	Routing Protocol Daemon	Maintains routing protocol functionality covering protocols such as BGP, OSPF, and RIP
`chassisd`	Chassis Daemon	Controls the chassis and environment processes
`Snmpd`	SNMP Daemon	Allows SNMP servers to interact with the device and allows reporting from the SNMP agent to server
`mgd`	Management Daemon	Controls management connections to the network device via SSH, telnet, or HTTPS
`dcd`	Device Control Daemon	Junos needs to communicate with the hardware and interface cards and the dcd permits this
`inetd`	Internet Service Daemon	Provides internetwork connectivity to Junos
`dhcpd`	DHCP Daemon	Allows Junos to offer dynamic IP addresses and perform DHCP discoveries

On the rare occasion when a daemon does stop responding to the system, it's not only important to know how to restart the daemon, but also how to identify the daemon that requires restarting.

The `restart` command allows an engineer to restart the individual daemons. This command needs to be followed with the name of the daemon you wish to restart. The name in this case, however, is the full name of the daemon, so instead of entering `chassisd`, the option `chassis-control` would need to be entered instead:

```
root@ACME-HQ-SRX-01> restart ?
Possible completions:
  802.1x-protocol-daemon  Port based Network Access Control
  application-identification  Application-identification process
  application-security  Application security daemon
  audit-process        Audit process
  autoinstallation     Autoinstallation process
  chassis-control      Chassis control process
  class-of-service     Class-of-service process
!
output omitted for brevity
!
  remote-operations    Remote operations process
  routing              Routing protocol process
  sampling             Traffic sampling control process
  sdk-service          SDK Service Daemon
  secure-neighbor-discovery  Secure Neighbor Discovery Protocol process
  security-intelligence  IPF daemon
  security-log         Security Log Daemon
  service-deployment   Service Deployment Client
  services             Restart a service
  simple-mail-client-service  Simple Mail Transfer Protocol Client process
  snmp                 Simple Network Management Protocol process
```

Now let's imagine for a moment that there is an issue with all routing protocols on your router and after some diagnosis you determine that the rpd daemon has crashed. As long as you know what rpd is short for, you can enter the restart command and use a question mark to display a list of daemons that can be restarted; at that point you'd need to simply find the option that closely matches rpd, which in this case is the option routing.

Once the correct daemon has been entered, there are three further options available:

- Gracefully – Attempts to restart the daemon gently without affecting other processes.
- Immediately – Kills and restarts the daemon without delay.
- Softly – Rereads and reactivates the configuration without completely restarting the software processes.

```
root@ACME-HQ-SRX-01> restart routing ?
Possible completions:
  <[Enter]>            Execute this command
  gracefully           Gracefully restart the process
  immediately          Immediately restart (SIGKILL) the process
  soft                 Soft reset (SIGHUP) the process
  |                    Pipe through a command
```

If no option is selected, the default option is to restart the daemon gracefully.

Once a daemon has been restarted using the `restart` command, an entry should appear in a log file. It should state that the daemon was terminated by signal number 9, followed by another message stating that the daemon has started:

```
Apr 12 18:13:46  ACME-HQ-SRX-01 init: routing (PID 2216) terminated by signal number 9!
Apr 12 18:13:46  ACME-HQ-SRX-01 init: routing (PID 2823) started
```

On a side note, you may notice that both log entries contain the letters PID and a number in parenthesis. PID is short for *process ID* and the number is the process ID number that was listed in the first column when we ran the `show system processes` command. If the same command is run once more but a filter is applied so that only the PID mentioned in the log, 2823, is displayed, the daemon rpd is shown in the output:

```
root@ACME-HQ-SRX-01> show system processes | match 2823
 2823  ??  S      0:06.39 /usr/sbin/rpd -N
```

If you refer to Table 1.1 once more, you can see that rpd is the daemon "routing protocol daemon". This is confirmation that when the command `restart routing` was run, it did indeed restart the daemon rpd.

Summary

By now you should have a better understanding of the fundamentals of the Junos network operating system, the architecture of Junos and the underlying FreeBSD operating system, and how to identify and restart daemons. You should also have a basic understanding of the Juniper family of network devices. Depending on when you are reading these pages, always refer to the Juniper web site for the latest product changes, enhancements, and specifications.

Chapter 2 discusses the user interfaces that are available to network engineers to configure and manage network devices running Junos.

Chapter 2

User Interface

By Chris Parker

I remember the first time I saw an enterprise-grade router. I was at the very beginning of my journey into networking, as some readers are now, entering a mysterious industry that was so alien to me that I barely even knew what an IP address was. I had a lot to learn, but I don't mind admitting that I was ridiculously excited to learn the difference between a router and a switch, a packet and a frame, a full night's sleep and being woken up at two in the morning because the council thought it would be a good idea to drill directly into some fiber-optic cables in the ground.

I noticed that this large, grey, loud enterprise-grade router was a far cry from the kind I had at home – the cheap and cheerful plastic box that came pre-configured by my ISP. My colleague told me they were going to somehow configure this blank router so it could talk on the network. And of course, the question that immediately sprang to mind was: How on earth does one go about configuring it? *How do you actually get onto these routers to make changes?*

I could see a number of ports that cables could potentially plug into on both the front and the back, and perhaps one of those cables could go from the router to my computer – but all the ports looked the same to me. And besides, even if I could get a cable from my computer to the router, what then? How would I actually log on to the box to manage it? I was at a loss for ideas. Via a web portal maybe? Perhaps there's a program I need to run on my machine to talk with the router? Do I put a disk in it? Do you shake it until it works? Do you have to start a conversation with the router, and hope it sees reason? Do I have to take it to dinner and attempt to win the router's respect?

Luckily, you don't have to do any of those things. Any romance you choose to initiate between you and your handsome Junos box is entirely optional. Let's learn about the real methods we can use to manage, log on, and successfully start breaking our Juniper routers, switches, and firewalls things. Er, wait: not breaking. I mean configuring. Configuring is what I mean.

How to Manage Juniper Boxes? An Introduction to the Functionality of the CLI

Almost every business-grade networking device currently on the market offers its power users – that's you! – a command-line interface, or CLI, as its primary method for interacting with the box. This CLI does what it says on the tin: it gives you a text prompt into which you can type commands and get responses.

Across all vendors, this CLI often gives you access to an almost unimaginable number of commands that allow you to display the current status of the box, as well as configure it, and to troubleshoot it when it breaks. And of course, the Junos OS is no exception. That's why the vast majority of this chapter is dedicated to introducing you to the power of the Junos CLI. We'll teach you the philosophy of the command line, and in doing so we'll set the scene for upcoming chapters where you'll learn to configure and troubleshoot a variety of very useful protocols, directly on the CLI itself.

In addition to the CLI, vendors sometimes offer a graphical user interface, or GUI. Some vendors offer it across all their devices, others only offer it on hardware designed for small-office/home-office scenarios to give less experienced users easy access to the box. This is great for small sites, where a business may want the advantages of enterprise-grade networking hardware even though the network itself may not be large enough to warrant employing a full-time engineer who is fluent in the unique syntax of their vendor of choice.

This book is mostly focused on the CLI. However, your real-world experience will feature a little bit of J-Web, which is Juniper's brand name for the graphical interface of Junos. At the end of this chapter we'll explore the J-Web, and see what functionality it offers.

Why have engineers historically preferred using a CLI over a GUI? Is it because the sci-fi-esq stream of green-on-black text running down their screen makes them feel like they're an extra in The Matrix? The answer is yes. Unquestionably, yes. However, there are also genuine advantages to using a command line instead of a graphical interface, such as the sheer speed at which it allows you to operate. Once you're familiar with even some basic commands, in seconds you can not only display a huge variety of live stats about the box you're on, but also easily compare them with historical stats, or the stats on other boxes. You can save these stats, import them, or run scripts on them to automate tasks.

In addition, it's possible to introduce automation directly via the CLI. For example, Junos allows you to create what it calls op scripts. Op is short for operational, and these scripts give you the ability to do things like create brand new custom commands that run a number of commands all at once. As you advance in your love affair with Junos you'll discover that you can, for example, bundle up a dozen or so troubleshooting commands utilized frequently by your users, and write a

script that deploys all of them with just one command – and then get Junos to not only print the results on the CLI, but also write them to a text file on the box. Then perhaps the script could even export that text file somewhere externally via FTP, so the results can be analyzed by anyone. It's also possible to do something like that via a graphical interface – but it's a whole lot easier on the command line!

I Heard That Command Lines Are Being Replaced With Automation. Is That True?

Yes. And no. To explain what I mean, let's talk for a moment about some different methods of automation.

Traditionally, the ethos of network engineers is to do everything on the CLI, and to very rarely use the GUI at all. Some engineers even turn the GUI off of their production boxes. This is partly because the CLI is generally better in terms of the functionality it offers, partly because nerds want to feel superior to people who like the GUI, and partly because it's easier to log what users do on the CLI than it is to log what they do on a GUI.

Some engineers also argue – incorrectly, in my personal opinion – that GUIs are a security risk. Their argument is that anyone can guess their way through a GUI: by contrast, it takes skill to operate a CLI. Turn off the GUI, and you turn off the ability for *n00b-hackers* with no CLI skills to still send five million viruses to your mainframe. My personal response to that would be: if your security is so weak that someone untrustworthy has gained access to your box's username and password, then GUI access is the least of your concerns.

Personally, I like both the CLI and the GUI, and I find advantages of each depending on the scenario. There is no right or wrong answer on how to configure a box – the tool that you personally find to be the quickest and easiest to use is the correct tool you should use for the job. The tool that allows you to most efficiently monitor and manage what people can and can't do is the correct choice. Sometimes that's a CLI, sometimes that's a GUI.

Having said that, we're hearing talk that the industry as a whole wants engineers to come off the CLI and either return to graphical interfaces that can manage an entire estate of boxes, or use automated scripts to configure individual boxes—or indeed an entire data center's worth of boxes.

Let's talk about these other GUIs first, because they're not necessarily the same as the GUI that the vendor offers with its hardware. Instead, these GUIs are driven by behind-the-scenes scripts that allow consistency in repetitive tasks. For example, imagine a large data center that frequently needs to deploy new switches to racks around the network. This data center uses the same model of switch at the top of every rack, and has a very consistent configuration on all of them. Ports 1 to 38

might all be given to customers, while the rest are used to connect the switch to the rest of the infrastructure. The only differences from switch to switch are the specifics: VLANs, IP addresses for gateways, that kind of thing.

In the past, these switches would be configured by the CLI. At best, an engineer might have a text file with a template configuration that they can paste the specifics, such as VLANs and IP addresses, into, but the possibility for human error in these scenarios is high. There's no verification when you're just pasting into a text file. I can attest to the pain of accidentally configuring a /24 LAN interface with a /32 subnet mask – in other words, a single IP address. Believe me when I say that I learned that lesson the hard way!

By contrast, engineers could type in the relevant VLANs and management IP addresses – all mandatory fields —using a built-in-house web interface so that nothing is forgotten, and not only have a script create an entire configuration, but automatically validate and deploy it onto a switch already connected to a dedicated implementation rack. This configuration will be guaranteed to have a consistent convention for interface names and descriptions, and not have a single VLAN out of place. The engineer can then take this switch and rank it into the location of choice. A large number of service providers work exactly like this, using GUIs that enable less technical staff to still accurately do things like configure new customer ports within their infrastructure.

As for writing scripts, this exact same scenario is possible by using automation tools like Ansible, Puppet, or Chef. For now it isn't too important to know what these are, or the differences between them; just know that they exist, and they allow you to write code that can also achieve the goal of automatically configuring and deploying networking devices. Using these tools, for example, a switch could acquire an IP by DHCP, call out to a particular server in the network, report its serial number, and pull down an automatically-created configuration based on the scripts you've written and its apparent location in the network.

Does making an in-house tool sound like too much fuss? It's common for larger companies to create such tools for themselves, but it's hardly scalable for smaller organizations. That's why vendors like Juniper offer a service where customers can plug a router or switch directly out of the box into the correct location of choice, at which point the device finds a default gateway, calls home to Juniper, again reports its serial number, and automatically gets a configuration created in advance by a company's engineers – all with zero touch from the engineer. Indeed, this is the essence of SD-WAN and SD-LAN technology, deployed with *zero-touch provisioning*. Both large and small organizations are increasingly taking advantage of this new paradigm.

Not only is the initial configuration slowly being moved to graphical interfaces that offer a consistent configuration experience, even daily operations are moving to GUIs. Ever-advanced point-and-click monitoring tools allow us to display

graphs and stats in more user-friendly ways, and potentially leverage the power of machine learning (like Juniper's Marvis technology, present in its next-generation Mist Wi-Fi offering) to automatically fix problems as soon as they occur.

As time goes on engineers are encouraged to use the CLI less and less. But less doesn't mean never. Ultimately, networks still break. The industry will always need people who truly understand the underlying protocols, the underlay that powers this automated overlay. Businesses will always need engineers who know how to spot when things have gone wrong.

And one thing will always be true: you can't automate what you don't understand. Almost all of these automation tools still require you to understand how the box works, and mastering the CLI is the very best way to learn how things really work under the hood. You can be sure that even if one day in the far future we truly managed to free ourselves from the CLI, we'll definitely still be using it for the foreseeable future.

So, yes, the CLI will certainly be used less than it was in the past. Considerably less. But don't believe anyone who says you'll never use it. And in any case, at the start of your career you don't need to worry too much about any of that. We chose to address this issue early in the book simply to remove it from your mind as a source of worry, because we know all too well what it's like to have senior engineers pressuring you into just believing as fact whatever opinions they tell you. Trust us: by learning the CLI, you will learn networking.

Having said that, never trust any engineer who insists that you listen to them and only them. The industry is in a constant state of change, and every engineer has their own opinion. Ask your (nicer) senior colleagues what they think. Explore blog posts on the issue of CLI versus GUI, and the issue of CLI versus automation. Listen to industry podcasts and read around on the topic. If you read this book and others, too, you'll become a network engineer.

The Interfaces Used to Manage Your Box

In Chapter 1, Martin Brown introduced you to some different models of Juniper devices, including SRX firewalls. Let's take a look at a photo of an SRX320, commonly found in smaller networks, shown in Figure 2.1.

There's a fair few interfaces on this fine beast. Notice that there's six physical Ethernet interfaces built into the chassis. Perhaps you can see that underneath they're labelled 0/0 to 0/5 These are Gigabit Ethernet interfaces. Generally in networking parlance, FastEthernet refers to 100 megabits (Mb) a second and Gig-Ethernet refers to speeds of 1,000Mb a second. (Though if we're being perfectly accurate it's actually 1,024Mb a second. But that's a story for another day!)

Figure 2.1 *An SRX320*

NOTE By the way, do you know the difference between megabits and megabytes? The details of the difference is out of scope for this book, but here are the highlights: a byte is made up of eight bits – meaning eight 1s and 0s, eight high or low voltages going across a wire that represent data. We use megabytes/gigabytes etc. to describe storage, but megabits/gigabits to describe network speed. It's very important to understand the difference. If someone says to you "I've got an 80 megabyte connection but I can only download at just under 10 megabytes a second," chances are they're not quite correct: they probably have an 80 megaBIT connection, not an 80 megaBYTE connection.

In addition to those six Gig-Ethernet ports that are built-in to the box, on the right of Figure 2.1 you can see two empty slots labeled 0/6 and 0/7. These are for SFPs, or small form-factor pluggable interface modules. These allow users to add a wide variety of interfaces, such as fiber-optic ports. Vendors tend to leave these SFP ports empty, allowing you to buy the module that suits your network.

To the left of the box in Figure 2.1 there's two interfaces labeled Console. We'll come back to those.

Finally, there's two expandable slots on the front of the chassis. The name for these slots actually changes between different models of Juniper hardware, which can get a little bit confusing sometimes. On a small SRX it's called a Mini-PIM, which stands for Physical Interface Module. You can buy cards that slot in here to give you extra interfaces, such as ADSL or Wi-Fi cards.

Can we use any these interfaces to manage the box, you might ask. We sure can! We can give them an IP address and then use that IP addresses to log on. But as you'll see in a moment, the console port is special – it lets us connect without even configuring an IP address!

Method One: Graphical Access Via J-Web

You may have heard of DHCP before. It stands for Dynamic Host Configuration Protocol, though it's not too important yet to memorize what it stands for. For now, just know that DHCP is what a computer uses to broadcast to the network that it needs an IP address, along with the other things it needs to operate, such as a subnet mask (how big the network's IP address space is), a default gateway (the IP address of a router that can give us access to things outside of the local network), and so on.

It's the job of DHCP servers to respond to these requests, and to offer IP addresses. Smaller branch firewalls like the SRX320 are set up out of the box to act as DHCP servers, which means we can connect our computer directly into one of those Ethernet ports, get an IP automatically, and start talking with the SRX over the web GUI.

Let's imagine that our SRX320 is taken out of its cardboard box and turned on for the very first time. If you were to configure your computer to get an IP address via DHCP, and if you were to then connect a cable from your computer to any of an SRX320's ports from 0/1 up to 0/6, your computer automatically receives an IP in the 192.168.1.0/24 range – that's 192.168.1.*anything*.

The SRX itself is configured as 192.168.1.1, which means we'd use that IP to connect to the SRX. As such, when you're using this special DHCP functionality, you log on to the box via the graphical J-Web interface by navigating to https://192.168.1.1 in your web browser of choice.

We'll talk more about the GUI at the end of this chapter, but for our current purposes know that you can either stay in the GUI, or if you're a pro – which you are – then you can set a password on the box, log off the GUI, and then carry on configuring the box via the CLI. How? Let's find out.

Method Two: SSH and Telnet

We're still plugged in to the same port on the SRX, and we've still got our DHCP IP address. We've briefly connected via J-Web, and set a root password on the box. By logging in as root, you are what's known as the super user. You have access to absolutely every command on the box.

Because we're committed to best practices, let's imagine that we additionally configured a second user account while we were there, so that we're not always using the root account. It's possible to make some users read-only, some users read-write, and even to make custom users that have a restricted view and control of the box based on your own requirements. You'll learn how to make these users in Chapter 3.

For now, let's imagine we made a user called *admin* with a nice strong password, like the kinds our grandfathers taught us about.

Now we've actually configured a username and password on the box, there are two protocols that allow us to connect to the command line of the router via an IP address: SSH, and Telnet. Think of both of these protocols as a kind of remote desktop for the command line. They both achieve the same thing – allowing us to operate and configure a box via the CLI over a remote connection – but there's a crucial difference: Telnet traffic goes across the wire in plain text. SSH traffic is fully encrypted. For this reason, try to use SSH whenever you can.

TIP Telnet works over TCP port 23, whereas SSH is TCP port 22.

If you're a Mac or Linux user, both of these protocols are built into the command line that comes with your computer. Open up a new terminal window (on Mac, go Finder > Applications > Utilities > Terminal, or simply hit CMD-Space and type Terminal), then, when your command line appears, do either of these two things:

Type: `telnet 192.168.1.1`, and enter `admin` as the username when prompted.

Type: `ssh admin@192.168.1.1`.

Whichever one you choose, type in the password when prompted. Your output should look something like this:

```
Chris-Parkers-MacBook:~ chrisparker$ ssh admin@192.168.1.1
Password:
--- JUNOS 19.4R1.10 Kernel 64-bit XEN JNPR-11.0-20191115.14c2ad5_buil
admin@Router1>
```

Congratulations—you've connected to the CLI of your Juniper SRX320!

What about Windows? On many versions there's no Telnet or SSH option built-in. Write to Bill Gates and ask him why.

Luckily there's a large number of applications you can download for free (or inexpensively) to give you this functionality. We favor the light simplicity of PuTTY, but other popular free and paid clients at the time of this writing include (but are not limited to) SecureCRT, MremoteNG, MobaXterm, KiTTY, and ZOC Terminal. (Apparently it's the law that all Windows command-line clients need to have ridiculous names.) Every network engineer has their own preferred Windows client, and every network engineer will tell you that their personal choice is the best, and that all other clients are terrible. We recommend that you take the advice of your colleagues with a pinch of salt, research all of them yourself, and see which one you prefer.

If you choose to use PuTTY, you'll see a screen like the one in Figure 2.2 when you open it. Notice that there's a space to put an IP address, and the option to choose Telnet or SSH. You can even save the session if you find yourself coming back to the same device regularly.

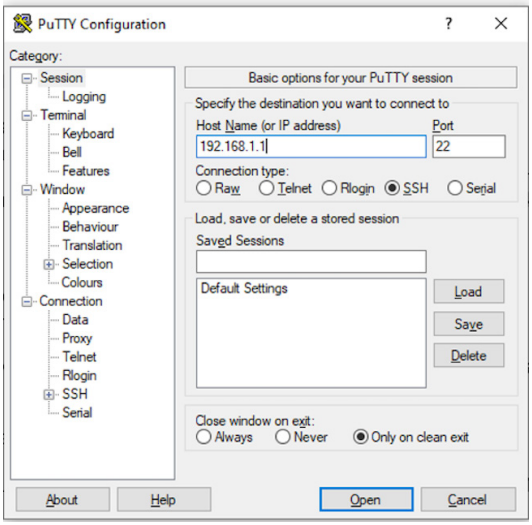


Figure 2.2 *PuTTY Screen*

Whatever client you use, and whether on Windows or Mac, when you SSH to the box you should see a prompt like this:

```
admin@Router1>
```

When you see the > symbol it means you're in something called operational mode. In a moment we'll talk about what that means, and we'll learn what we can actually do now that we're there. First, though, let's talk about two other important methods of connecting to a Juniper box.

Method Three: The Console Port

In the previous example our computer acquired an IP from the SRX320 via DHCP. This almost always works on smaller branch devices. However, the larger chassis that you'll find in data center and service provider networks, like a larger SRX firewall or an MX router, actually come with all interfaces completely blank! No IP address, no DHCP, nothing.

Why is this? In a smaller enterprise LAN environment, there's a strong chance that the office will only have one single, simple, flat LAN. There's also a strong chance that there won't be any dedicated network engineers on site. For both of these reasons, offering a preconfigured LAN with DHCP offers an excellent advantage when it comes to ease of deployment. Small offices can get up and running very quickly, with the option of hopping onto the J-Web if they need to customize anything.

By contrast, larger boxes come with any number of interfaces, and no vendor can assume that any two companies will use a particular box to perform the same tasks. Every network is unique and beautiful in its own way. For example, look at some of the different models of an MX router in Figure 2.3.

Figure 2.3 *The Beautiful MX Series Family*

Figure 2.3 emphasizes that these aren't one-size-fits-all boxes. They're certainly not used in small enterprises with a single flat LAN and without network engineers! Even the smallest box, the MX5, packs a very powerful punch indeed. For that reason, all interfaces on these larger boxes come completely blank and unconfigured, save for a few default physical settings.

This begs the question: *if there's no IP addresses on these boxes, how do we log on to them?*

Go back to Figure 2.1 of the SRX and look to the left of the 0/0 interface. Do you notice that there are two interfaces labelled Console? One of them looks like an Ethernet port, and one of them looks like a micro-USB port. Both act as what network engineers call either the *console* port or the *serial port*, depending on what mood we're in that day.

By connecting to the console port, it's almost as if you're making a direct connection into the router itself. There are no IP addresses involved – your computer is directly connecting into the heart and soul of the device, to talk intimately with it. This is very useful: network engineers regularly console directly into devices when they're brand new. We also console in if the box appears to have crashed, in the hope that we might be able to gain access through this direct connection when Telnet or SSH fails for some reason.

Of the two console interfaces, the *Ethernet-shaped one* – more accurately described as an *RJ45 presentation* – takes a special kind of cable called a *console cable*. Sometimes you'll also hear it called a *rollover cable*. There should be one

included in the shipping box of your Junos device. Alternatively, you can buy them online, literally for pennies. One end of the console cable plugs into this port. Nowadays, the other end most commonly goes into the USB port on your computer. Some very old computers also have what are called *serial ports*, which are chunky but small interfaces that are about an inch wide, with nine pins in them.

Macs and Linux boxes have a particular command to allow you to use your Terminal app to access a USB console connection. You'll need to check the help files for your particular operating system for the specifics, but it usually involves typing `cd /dev` and then searching this folder for a file (really an interface) that contains the text USB or ACM. You'll then type something like `screen /dev/tty.YOUR_USB_INTERFACE 9600`, replacing that with whatever your interface was called. (There's also a chance you'll need to install some extra drivers on your machine to make this work.)

If you're using PuTTY, you simply have to choose Serial as your connection type. The Serial line section should change to COM1. If you get a connection error, try changing COM1 to COM2, COM3, or COM4.

However you do it, if it worked you should be faced with a blank screen. Press the return key, and you should then see a login prompt!

On the router itself, the RJ45 console port is the most commonly used port. The micro-USB port works too, though you may have to install more drivers on your machine (available via the Juniper website) to make it work, and you may possibly have to add some extra configuration via the CLI to enable it – meaning you'll still need to console in via the RJ45 port first.

NOTE At this stage, other study guides will bore you with settings like the *baud rate*, the *parity,* and the *stop bits*. In practice it's rare that you ever need to touch these settings, so we'll skip over them for now. If you ever find yourself having to play with these, check Juniper's official documentation for advice at https://www.juniper.net/documentation.

In the next output we're consoling on to a brand new router that's never received a single piece of configuration. Type your username of `root`. By default there's no password, meaning you should be taken directly in, and given a prompt like this:

```
Amnesiac (ttyd0)

login: root

--- JUNOS 14.1R1.10 built 2014-06-07 09:37:07 UTC

root@%
```

When you see `Amnesiac` in your output, you know you're logging on to a brand-new device (only the root user gets into the UNIX shell).

Did you notice that, unlike with our SSH output, when we console in we don't have a > symbol? Instead we have a % sign. This is important: when you console in, you're not actually taken into operational mode. Instead, you're put directly into the Unix shell! In Chapter 1 you learned that Junos actually runs on top of a flavor of Unix called FreeBSD, or more recently, Linux. When you console in, you're taken directly into this FreeBSD shell. (You can also get to it from operational mode by typing `start shell`.)

As you step up in your career, you will find occasion to be in this shell. There are some advanced troubleshooting commands that can only be run from here, and they're usually commands that Juniper's technical support team, JTAC, will ask you to run. When you find yourself in this shell, you can type `cli` to enter back into operational mode, and that's what we're interested in:

```
root@% cli
root>
```

Perfect! There's one final way to connect to the box. This method also uses Telnet or SSH – but there's something unique about it.

Method Four: The Management Port

On some Juniper boxes you'll find an interface labelled MGMT. This is known as the *management interface*. When you use the CLI or J-Web to look at the actual list of interfaces on the box, it's usually referred to as the fxp0 interface, though on some boxes you'll see it called em0 instead.

You configure this interface to have an IP address, just like any other interface but the difference is that this interface doesn't allow traffic to transit it. In other words, you can get to the router itself via this interface but you cannot get to the rest of your network via this interface. And similarly, nothing can get out of your network via this interface.

This interface is also sometimes called the *out-of-band interface*, because it usually hosts a direct connection to an ISP. Technically, this connection lives outside of your network – out of band, if you will – and thus, if you're trying to manage the box remotely, this interface gives you a direct connection into the box that won't be affected by any outage or misconfiguration within your network. If you choose to use this interface, it gives you the ability to Telnet or SSH to the box even if your entire network is down. As long as your out-of-band link is up, you'll be able to connect to the box and troubleshoot the problem.

NOTE Technically speaking, your console port is also *out-of-band*, which can lead to some maddening inconsistency in literature across all vendors. We've seen both the console and the MGMT interface described as out-of-band interfaces. If someone ever talks to you about the OOB interface, and you're unsure which interface they're referring to, don't be afraid to ask for clarity!

Operational Mode: The First of Two CLI Modes

Whether you used SSH, Telnet, or the console port with the `cli` command, you're now successfully connected to your Juniper box. Congratulations: a new world of heart-stopping late-night core network changes awaits you!

A moment ago we mentioned operational mode. This is the mode where you'll find yourself typing *operational commands*, which give you the ability to manage and troubleshoot the box. Some of these operational commands give you a snapshot of information in the moment you type them, while others allow you to view traffic and connectivity in real-time.

By contrast, later on we'll learn about *configuration mode*, where you can actually make changes to the box as well as running debugging commands called *traceoptions*, which offer more in-depth, processor-intensive information about how various parts of the box are operating. Configuration mode is the place where you can actually do damage to the box. Operational mode commands *tend* to be safe, though there are a few notable exceptions, such as the `request system reboot` command. Can you guess what that command does? Give yourself ten points if you can.

Many of the commands you'll use in operational mode begin with the word `show`. There's a `show` command for almost anything you could possibly want to learn regarding the operational status of your box, and my colleagues and I will be showing you plenty of them throughout this book. Let's start simple, and check what model of box we're on, using the `show version` command :

```
root> show version
Hostname: Router_3_Edge
Model: vmx
Junos: 14.1R1.10
JUNOS Base OS Software Suite [14.1R1.10]
 {snip}
```

Neat! As you can see, we're currently working on a virtual MX router, as shown by `Model: vmx`. However, all of these commands you're going to learn in this section work exactly the same whether the box is virtual or physical – and indeed, as Martin mentioned in Chapter 1, all these commands are precisely the same regardless of whether you're on a router, a switch, or a firewall.

This is one of the wonderful things about working on Junos boxes. Some other vendors have totally different syntax between different models of hardware, forcing you to learn five different versions of the same command. By contrast, there is one Junos across Juniper's entire product set. The only differences come between boxes that have different features. For example, firewalls will have special security functionality not available on a switch, so certain security commands won't be available

on non-security products. Similarly, some special switching functionality won't be available on firewalls. But for the commands that the products share, the commands are the same.

Actually, there have been times where the commands have differed between products for historical or architectural reasons. But whenever that happens, efforts are then made to synchronize the syntax in later versions of Junos. For example, for a short time there was a difference between the way you configured VLANs on MX routers and EX switches. In the spirit of One Junos, EX switches were altered to use what Juniper called ELS, or Enhanced Layer-2 Software. This new syntax gave consistency across Juniper's full Junos-powered product set.

It's important to know that ELS exists. Knowing about ELS will explain why you see slightly different configuration on older boxes and in older blog posts you'll stumble across in your studies, compared to the config you might see in newer documentation. For now, just be aware that some older boxes may sometimes be configured slightly differently from the way you've learned it should work, and that a quick internet search for Junos ELS should give you clarity.

Almost all of the output examples you're going to see in this chapter come from the virtual MX in my personal lab. The configuration on this vMX is very small, and there are only a few interfaces, so the output won't be too overwhelming. However, sometimes it will be useful to show you output from a real production router with more traffic going through it, to see a richer variety of statistics. You'll be able to spot this because the hostname of the box will show as `chris.parker@Production_Box>`.

So, we know what kind of box we're on. By the way, what's the time now? And how long has this box been up?

```
root> show system uptime
Current time: 2019-12-18 17:27:51 UTC
System booted: 2019-12-18 17:22:51 UTC (00:05:00 ago)
Protocols started: 2019-12-18 17:23:32 UTC (00:04:19 ago)
Last configured: 2019-12-18 17:23:28 UTC (00:04:23 ago) by root
 5:27PM  up 5 mins, 1 user, load averages: 0.14, 0.48, 0.29
```

The box has been up for five minutes because I just spun it up a moment ago. Whenever customers call in with a problem and they claim to have rebooted their Juniper device, I always run this command to check whether they're telling the truth. I certainly believe them when they say they've rebooted something – I just don't necessarily believe that they rebooted the correct thing!

Notice that `show system uptime` also shows you the current time on the box. Make sure it's always correct! If your time is wrong, your time-stamped logs will be a nightmare to read. In later chapters of this book we'll show you how to adjust the time on your box.

Navigating Around Operational Mode: Command Shortcuts

If you've come from other vendors, you might know that instead of typing `show version`, you could just type `sh ver`, and it automatically completes the command for you. This is a nice time saver, or tim-sav as I like to call it.

For example, if the first word of your command is `show`, and there are no other commands whose first word starts with `sh`, the boxes of most vendors are smart enough to let you just type `sh` when you mean `show`. This greatly improves the speed at which you can type commands, and therefore go home.

There are two ways to auto-complete your commands: the space bar, and the tab key. This is true on almost all vendors, though annoyingly most vendors actually only auto-complete the word that's displayed on the screen when you use the tab key. For some reason, other vendors decided to leave `sh ver` on the screen when you use the space bar – which looks very messy when you want to share your work with colleagues, or on internal knowledge base posts, or in blog posts, or when you find yourself writing networking books.

By contrast, regardless of whether you press tab or space, your Junos box auto-completes the output to display the full command. This is far more clear, far more tidy, far more readable by everyone! Check it out: I just typed `sh sys up` on my CLI. Every time I pressed space, here's what it automatically completed to:

```
root> show system uptime
```

Perfect! You'll never find a Junos command history that simply says `sh sys up`: Junos tidies things up for you as you go.

(There's an extra advantage to using the tab button, though. Later on you're going to learn how you can make your own routing policies and firewall filters. You can name these whatever you want. The space bar automatically completes only built-in commands. By contrast, the tab button automatically-completes even your own user-defined objects.)

What happens if you half-type a word, but it turns out that there are multiple things that it could translate to? Luckily, the moment you hit space or tab, Junos helpfully offers you this output, so you can choose for yourself what you mean:

```
root> s
      ^
's' is ambiguous.
Possible completions:
  save                 Save information to file
  set                  Set CLI properties, date/time, craft interface message
  show                 Show system information
  ssh                  Start secure shell on another host
  start                Start shell
```

Navigating Around Operational Mode: An Introduction to the CLI Hierarchy

Did you notice earlier that the command we typed wasn't `show uptime`, but `show system uptime`? There's actually a very predictable hierarchy to the commands within Junos – and understanding this hierarchy makes guessing your way to finding the command you need extremely easy.

As the command goes from left to right, it gets more and more specific. For example, typing `show interfaces` simply shows you stats about every interface on the box. By contrast, `show interface diagnostics optics ge-0/0/0` gives you information specifically about fiber-optic stats, and more specifically just for interface ge-0/0/0.

To see this hierarchy in action, let's type the word `show` followed by a question mark. You don't need to press return: just typing the ? symbol is enough. Doing this immediately prompts Junos to show you all the possible commands available to you:

```
root> show ?
Possible completions:
  accounting           Show accounting profiles and records
  amt                  Show AMT Protocol information
  ancp                 Show ancp information
  app-engine           Show App-engine information
  aps                  Show Automatic Protection Switching information
  arp                  Show system Address Resolution Protocol table entries
  as-path              Show table of known autonomous system paths
  backup-selection     Show backup selection policies information
  bfd                  Show Bidirectional Forwarding Detection information
  bgp                  Show Border Gateway Protocol information
  bridge               Show bridging information
  chassis              Show chassis information
  class-of-service     Show class-of-service (CoS) information
  cli                  Show command-line interface settings
  configuration        Show current configuration
  connections          Show circuit cross-connect connections
  database-replication  Show database replication information
  ddos-protection      Show DDOS information
  dhcp                 Show Dynamic Host Configuration Protocol information
  dhcp-security
  dhcpv6               Show Dynamic Host Configuration Protocol v6 information
  diameter             Show diameter information
---(more 23%)---
```

Wow, that's a lot of show commands! And that's only screen one of many. The `---(more 23%)---` at the bottom means that my screen is only showing me 23% of the total output. At this point I could press space to see the next screen, or I could press the return key to just go down one line at a time.

In any case, you can see that there's an awful lot of show commands available to us. Some commands that you'll come to learn in future chapters include `show arp` to see what devices your router has discovered on a local Ethernet segment, `show`

`configuration`, which we'll use later in this chapter, and `show dhcp`, which is the start of a number of commands that show you DHCP-related information. In other words, show is at the top of the hierarchy, DHCP comes next, and then after that, you get these options:

```
root> show dhcp ?
Possible completions:
  client               Show DHCP client information
  proxy-client         Show DHCP proxy-client information
  relay                Show DHCP relay information
  server               Show DHCP server information
  statistics           Show DHCP service statistics
root>
```

Perhaps it's a little bit clearer now why you don't need to memorize every single command. Once you understand the hierarchy, you can use the question mark key to navigate even to previously-unexplored areas of your Junos box!

What if we delete the word `show`, and just type a question mark? Will we be able to see what commands are available to us in operational mode other than `show`? You bet we can!

```
root> ?
Possible completions:
  clear                Clear information in the system
  configure            Manipulate software configuration information
  file                 Perform file operations
  help                 Provide help information
  load                 Load information from file
  monitor              Show real-time debugging information
  mtrace               Trace multicast path from source to receiver
  op                   Invoke an operation script
  ping                 Ping remote target
  quit                 Exit the management session
  request              Make system-level requests
  restart              Restart software process
  save                 Save information to file
  set                  Set CLI properties, date/time, craft interface message
  show                 Show system information
  ssh                  Start secure shell on another host
  start                Start shell
  telnet               Telnet to another host
  test                 Perform diagnostic debugging
  traceroute           Trace route to remote host
root>
```

On this list you can see some troubleshooting tools like `ping` and `traceroute`. You may know already that we can use `ping` to see if other boxes in the network reply to us, and we can use `traceroute` to see all the routers in between us and someone else. Don't worry if those commands are new to you: later chapters of this book will give you an introduction to troubleshooting your network.

There's `configure`, which we'll definitely be using later on, and `request`, which is the start of the `request system reboot` command we learned earlier. Once again, in that

command we see this hierarchy at play; Juniper could have just used `reboot` as the command to reboot the box – but a little bit of extra typing leads to a highly predictable command structure.

Junos offers a great option above some other vendors, you can use ? even to see items that you've named yourself. For example, if you wanted to view stats relating to a certain firewall filter, you could type `show firewall counter filter ?` to see a list of all the firewall filters you've configured. When you think about it, it's absurd that the boxes of other vendors don't give you this information!

Using the ? symbol as you type is a brilliant way to learn the CLI. You'll become familiar with the command hierarchy and what options are available to you very quickly, including options you might not otherwise have known existed. Going back to our `show system` commands, let's do a question mark after that, and see what options are available to us within this part of the hierarchy:

```
root> show system ?
Possible completions:
  alarms               Show system alarm status
  audit                Show file system MD5 hash and permissions
  boot-messages        Show boot time messages
  buffers              Show buffer statistics
  certificate          Show installed X509 certificates
  commit               Show pending commit requests (if any) and commit history
  configuration        Show configuration information
  connections          Show system connection activity
  core-dumps           Show system core files
  directory-usage      Show local directory information
  license              Show feature licenses information
  login                Show system login state
  memory               Show system memory usage
  processes            Show system process table
  queues               Show queue statistics
  reboot               Show any pending halt or reboot requests
  resource-cleanup     Show resource cleanup information
  rollback             Show rolled back configuration
  services             Show service applications information
  snapshot             Show snapshot information
  software             Show loaded JUNOS extensions
  statistics           Show statistics for protocol
  storage              Show local storage data
  subscriber-management  Show Subscriber management information
  uptime               Show time since system and processes started
  users                Show users who are currently logged in
  virtual-memory       Show kernel dynamic memory usage
```

Some commands that I frequently use in this show system hierarchy include:

- `show system storage`: shows disk usage across the various partitions of the internal disk.
- `show system license`: shows what licenses are installed on the box, and when they expire.

- `show system commit`: shows all the previous configurations stored on the box. Remember this one: we'll come back to it later.
- `show system users`: shows all the users that are currently logged onto the box.

There's a number of show commands you'll use, including `show interfaces`, `show route`, `show chassis routing-engine`, and more. In future chapters you'll learn all about them, and in a moment we'll learn a little about `show interfaces` in particular. First though, let's talk more about this idea of navigating around the CLI.

TIP If multiple users are logged on at once, and if you're sure they're not doing anything essential, why not give them a scare by typing the command `request message all message Hi everybody!`. This output will be sent to the CLI of all other users currently connected to the box. When used correctly, this is a brilliant way to inform everyone that you're about to start some important maintenance, or to warn people to save their work before an upcoming reboot. Alternatively, when you use this command for jokes, you can make people believe that their CLI is haunted. I leave it to you to decide your own best practices.

CLI Keyboard Shortcuts

The space bar and tab key offer great keyboard shortcuts to reduce your typing. Though as you read through this book, you'll notice that some commands can get very long indeed. What happens if you get to the end of the line, and decide you want to delete it all and start again? You could hit the backspace key and wait a long time, or you could just press Ctrl+u. This keyboard shortcut deletes the entire line all in one go!

Junos comes with a number of keyboard shortcuts like this. You might instinctively know that you can press up and down to scroll through your entire command history, but that's only the start of it. Great keyboard shortcuts that I find particularly useful include Ctrl+w, which deletes one word at a time. I often find myself pressing Ctrl+w a few times in quick succession, if I realized I've typed the second half of a command incorrectly. Ctrl+a and Ctrl+e are also very useful for quickly getting to the beginning and end of a line, respectively.

Table 2.1 is a list of every single keyboard shortcut, taken directly from Juniper documentation. Don't worry about memorizing them by row: instead, try each of them out on the CLI yourself. Make an effort to learn one a week, and to actively use it. You'll soon have them all burned into your memory for life, and you'll find yourself using them without even thinking.

If you're a Mac or Linux user, try these commands out on your computer's command line even when you're not on your Junos box. You'll find that most, and perhaps all, of them work in your Terminal app. That's not surprising: Junos runs on top of FreeBSD, after all!

Table 2.1 Junos Keyboard Shortcuts

Keyboard sequence	Action
Ctrl+b	Move the cursor back one character.
Esc+b or Alt+b	Move the cursor back one word.
Ctrl+f	Move the cursor forward one character.
Esc+f or Alt+f	Move the cursor forward one word.
Ctrl+a	Move the cursor to the beginning of the command line.
Ctrl+e	Move the cursor to the end of the command line.
Ctrl+h, Delete, or Backspace	Delete the character before the cursor.
Ctrl+d	Delete the character at the cursor.
Ctrl+k	Delete the all characters from the cursor to the end of the command line.
Ctrl+u or Ctrl+x	Delete the all characters from the command line.
Ctrl+w, Esc+Backspace, or Alt+Backspace	Delete the word before the cursor.
Esc+d or Alt+d	Delete the word after the cursor.
Ctrl+y	Insert the most recently deleted text at the cursor.
Ctrl+l	Redraw the current line.
Ctrl+p	Scroll backward through the list of recently executed commands.
Ctrl+n	Scroll forward through the list of recently executed commands.
Ctrl+r	Search the CLI history incrementally in reverse order for lines matching the search string.
Esc+/ or Alt+/	Search the CLI history for words for which the current word is a prefix.
Esc+. or Alt+	Scroll backward through the list of recently entered words in a command line.
Esc+number sequence or Alt+number sequence	Specify the number of times to execute a keyboard sequence.

Looking at Interfaces in Operational Mode

In Chapter 3 you're going to learn about the different kind of interfaces that a Junos device can utilize. Then, in Chapter 4, you'll take a deeper dive into the various show commands to view and troubleshoot these interfaces.

To set the scene for these chapters, let's take an introductory look at how you might view the operational status of an interface. Interfaces are a core component of network devices after all, and they offer us a brilliant opportunity to learn some new tricks on the CLI.

Typing `show interfaces` by itself gives you some stats about every single interface on the box. That's a bit too much information, so let's focus just on the stats for the ge-0/0/0 interface by typing `show interface ge-0/0/0`. Behind the scenes I've hopped into configuration mode and put an IP address on this interface, as well as naming my brand new router *Router1*. Later on I'll show you how to do this. For now, let's look at the result:

```
root@Router1> show interfaces ge-0/0/0
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 137, SNMP ifIndex: 514
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, Speed: 1000mbps, BPDU Error: None, MAC-
REWRITE Error: None, Loopback: Disabled, Source filtering: Disabled, Flow control: Enabled
  Pad to minimum frame size: Disabled
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x4000
  Link flags     : None
  CoS queues     : 8 supported, 8 maximum usable queues
  Current address: 00:05:86:71:cc:00, Hardware address: 00:05:86:71:cc:00
  Last flapped   : 2019-12-18 17:24:37 UTC (00:44:56 ago)
  Input rate     : 115416 bps (171 pps)
  Output rate    : 115584 bps (171 pps)
  Active alarms  : None
  Active defects : None
  Interface transmit statistics: Disabled

  Logical interface ge-0/0/0.0 (Index 329) (SNMP ifIndex 524)
    Flags: Up SNMP-Traps 0x4004000 Encapsulation: ENET2
    Input packets : 6233
    Output packets: 6252
    Protocol inet, MTU: 1500
      Flags: Sendbcast-pkt-to-re, Is-Primary
      Addresses, Flags: Is-Default Is-Preferred Is-Primary
        Destination: 10.10.12/24, Local: 10.10.12.1, Broadcast: 10.10.12.255
    Protocol multiservice, MTU: Unlimited
      Flags: Is-Primary
```

(You might have guessed that the ge bit in the interface name means gigabit-ethernet. The 0/0/0 bit is less intuitive: it refers to precisely where in the hardware this interface is located. On smaller boxes the interfaces will always be labeled something like 0/0/x. Bigger boxes with more line cards will have different numbers in these three slots, indicating the specific line cards the interfaces are physically installed on.)

As you can see from our example output, there's a lot of information on display – most of which we can ignore! Let's highlight a few things of particular interest. Read through the output, and see if you can spot:

- The fact that this is running at 1Gb (Speed: 1000mbps).
- The fact that the MAC address (i.e., the hardware address) is 00:05:86:71:cc:00.
- The date and time that the interface last went down (i.e., last flapped).
- The current average input and output rate, in both bytes per second and packets per second.

There's also some information about something called the logical interface. Some other vendors call this a *sub-interface*. You'll learn about what this means in detail in Chapter 3, but here's the concept in brief: every physical interface is *always* given at least one logical interface. The physical interface is written as ge-0/0/0, and you configure the physical properties on this interface. By contrast, ge-0/0/0.0 represents the logical interface. The .0 at the end is sometimes called a unit, as you'll see later on when we come to actually configure our first interface. It's on the unit that we configure IP addresses.

Physical interface = physical properties; logical interface = logical properties. Simple, right?

For now we'll just be working with physical interfaces that host one single unit, i.e., one single logical interface. However, as you'll learn in later chapters, it's perfectly possible, and indeed, very common, for one physical interface to host multiple logical interfaces. If you're not yet familiar with the concept of VLANs and VLAN tagging, then for now just be aware that one physical interface can connect to multiple LANs, using this concept of logical interfaces. The idea will become clearer as you progress through this book.

Terse, Brief, Detail, and Extensive: Getting More or Less from Your Command

We saw in that output that you could use the `show interfaces` command to see the bandwidth currently going through an interface. However, if you're used to other vendors, you might expect to see more information than just that. For example, where are the error counters to help you in your troubleshooting? Don't fear: not only can you get access to this information in Junos, but you can actually see far more than some vendors let you see.

To get the full output for an interface, type `show interfaces ge-0/0/0 detail`, or if you're brave, `show interfaces ge-0/0/0 extensive`. Equally, if you want to see less information, you can type `brief` or even `terse` at the end. These four commands – terse, brief, detail, and extensive – are available at the end of most Junos commands. As you get more comfortable on the CLI you'll find yourself using them all the time to focus in on the information you need in that moment.

Let's see what effect adding `terse` has on our output:

```
root@Router1> show interfaces ge-0/0/0 terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
                                  multiservice
```

This is really useful. You can very quickly see if an interface is up or down. You'll find yourself using this one a lot!

You can see a physical and a logical interface there. Notice that the IP address lives on the logical ge-0/0/0.0 interface. This is what the Protocol `inet` means. If this interface had an IPv6 address, you'd see a Protocol of inet6.

Notice as well that there are two columns in this output: Admin, and Link. Both are up, but what's the difference? It can be a little complex, so let's break it down.

On both the physical ge-0/0/0 and the logical ge-0/0/0.0 interface, the Admin column tells us whether or not the interface has been manually disabled by an administrator, via configuration. You often hear this being described as setting an interface to be *admin-down*. We haven't yet learned about configuration, so what I'm about to tell you next is a little peek into the future: you can shut down a physical interface with the configuration command `set interfaces ge-0/0/0 disable`. When I do that, notice how the output changes: the Admin and Link status of the physical interface is now down:

```
root@Router1> show interfaces ge-0/0/0 terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                down  down
ge-0/0/0.0              up    down inet     10.10.12.1/24
                                  multiservice
```

It might seem weird that the physical interface is Admin down while the logical interface remains up, but once you understand what this is telling you, it's actually really helpful: you now know what you need to do in the configuration to bring this interface up, because you can see that it's not the logical interface that's shut down: it's the physical interface.

To shut down one single logical interface, you would instead type `set interfaces ge-0/0/0 unit 0 disable`. Notice this time that we're shutting down unit 0 in particular. This will make the physical ge-0/0/0 show as up, and the logical .0 interface as down. Any other logical interfaces we add in the future will also remain up.

By contrast, again on the physical ge-0/0/0 interface, the Link column indicates whether the physical link is working. If this is down, it could indicate a bad cable, a hardware fault, or it could even indicate a duplex/speed mismatch between the two ports. You'll learn more about what this means in Chapter 3. As for the Link column on the logical interface, this will only be down if the physical interface also has the link as down.

The fact that we can spend more than one page simply talking about viewing the terse output of one interface is a testament to just how much power the CLI can give you in troubleshooting issues with your network.

Filtering Output

So now you know how to read the output of `show interface ge-0/0/0 terse`. Interestingly, you can also type `show interface terse ge-0/0/0`. On the one hand there is a strict hierarchy of how commands are found on Junos devices. On the other hand, there is often flexibility in the order in which you can type these commands. The words are in a different order, but there's a clear consistency. When you're awake at 2 a.m. trying to troubleshoot a problem, you'll come to be very grateful for this freedom!

What happens if we remove the ge-0/0/0 from the `show interfaces terse` command? You'll see the status of every physical and logical interface on the box. However, if you're coming from other vendor's equipment, you might be surprised by the output. Bearing in mind that I've assigned ten virtual gig-ethernet interfaces to my vMX, and bearing in mind that so far I've only configured the ge-0/0/0 and ge-0/0/1 interfaces, this is the entire output of my command:

```
root@Router1> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
                                  multiservice
lc-0/0/0                up    up
lc-0/0/0.32769          up    up   vpls
pfe-0/0/0               up    up
pfe-0/0/0.16383         up    up   inet
                                  inet6
pfh-0/0/0               up    up
pfh-0/0/0.16383         up    up   inet
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     10.10.13.1/24
                                  multiservice
ge-0/0/2                up    up
ge-0/0/3                up    up
ge-0/0/4                up    up
ge-0/0/5                up    up
ge-0/0/6                up    up
ge-0/0/7                up    up
ge-0/0/8                up    up
ge-0/0/9                up    up
cbp0                    up    up
demux0                  up    up
dsc                     up    up
gre                     up    up
ipip                    up    up
irb                     up    up
lo0                     up    up
lo0.16384               up    up   inet     127.0.0.1           --> 0/0
lo0.16385               up    up   inet     128.0.0.4           --> 0/0
```

```
                                    inet6    fe80::200:f:fc00:0
lo0.32768               up    up
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
pip0                    up    up
pp0                     up    up
tap                     up    up
vtep                    up    up
```

Wow, that's a lot of interfaces. What gives?

Junos, in an effort to be helpful, shows you the status of every single interface on the box, including virtual interfaces in its memory, and including interfaces you're not even using yet. This can be overwhelming even on a small SRX320, let alone a huge MX960. Luckily, there's a few things we can do to make it a bit more readable.

The Junos CLI provides a variety of options to filter out unwanted text, helping you to focus in on just the stuff you want to see. Put a pipe (|) at the end of any command, and hit that friendly question mark to see what you can do:

```
root@Router1> show interfaces terse | ?
Possible completions:
  append               Append output text to file
  count                Count occurrences
  display              Show additional kinds of information
  except               Show only text that does not match a pattern
  find                 Search for first occurrence of pattern
  hold                 Hold text without exiting the --More-- prompt
  last                 Display end of output only
  match                Show only text that matches a pattern
  no-more              Don't paginate output
  refresh              Refresh a continuous display of the command
  request              Make system-level requests
  resolve              Resolve IP addresses
  save                 Save output text to file
  tee                  Write to standard output and file
  trim                 Trim specified number of columns from start of line
```

Let's play with a few of these, starting with `match`. As you might imagine, `match` allows you to view only lines that contain a certain bit of text. Let's try looking only at lines that have `ge` in them. With any luck, this will filter out all interfaces apart from our Gig-Ethernets:

```
root@Router1> show interfaces terse | match ge
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     10.10.13.1/24
ge-0/0/2                up    up
ge-0/0/3                up    up
ge-0/0/4                up    up
ge-0/0/5                up    up
```

```
ge-0/0/6                up    up
ge-0/0/7                up    up
ge-0/0/8                up    up
ge-0/0/9                up    up
```

Indeed it does! And it doesn't end there – you can even do more pipes after that to match again. Let's imagine that we want to take this filtered output, and filter it further, showing us only the lines that have `inet` in them. In Junos it's a breeze:

```
root@Router1> show interfaces terse | match ge | match inet
ge-0/0/0.0              up    up   inet     10.10.12.1/24
ge-0/0/1.0              up    up   inet     10.10.13.1/24
```

Very clean!

That previous example finds every line that contains the text `ge`, and searches this filtered output to match only for lines that contain `inet`. This time, let's imagine that we wanted to view any lines that contain either the text `ge`, or the text `inet`. Is it possible? It sure is! Try this command:

```
root@Router1> show interfaces terse | match ge|inet
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
pfe-0/0/0.16383         up    up   inet
                                   inet6
pfh-0/0/0.16383         up    up   inet
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     10.10.13.1/24
ge-0/0/2                up    up
ge-0/0/3                up    up
ge-0/0/4                up    up
ge-0/0/5                up    up
ge-0/0/6                up    up
ge-0/0/7                up    up
ge-0/0/8                up    up
ge-0/0/9                up    up
lo0.16384               up    up   inet     127.0.0.1           --> 0/0
lo0.16385               up    up   inet     128.0.0.4           --> 0/0
                                   inet6    fe80::200:f:fc00:0
```

We just used what nerds call a *regular expression*. Our match condition searches for either `ge` or `inet` at the same time. On some vendor's OS, this kind of search is impossible. On Junos, you can do it in your sleep.

This match functionality becomes even more useful when you're using the `detail` and `extensive` options. The command `show interfaces ge-0/0/0 extensive` command gives so much information that including the full output in this book would take up three entire pages! Type it for yourself in your lab (and if you don't have a lab yet, don't worry – check the Appendix at the back of this book for information on how to set up your own free lab), and look at the output line by line. Do you notice that we suddenly see those error counters we mentioned earlier? That's great, but what if we want to quickly look at the error counters on ge-0/0/0, and when they last flapped? Perhaps we could use something like this:

```
root@Router1> show interfaces ge-0/0/0 extensive | match error|flap
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, Speed: 1000mbps, BPDU Error: None, MAC-REWRITE
Error: None, Loopback: Disabled, Source filtering: Disabled, Flow control: Enabled
  Last flapped   : 2019-12-18 18:21:48 UTC (00:12:38 ago)
  Input errors:
    Errors: 0, Drops: 0, Framing errors: 0, Runts: 0, Policed discards: 0, L3 incompletes: 0, L2 channel
errors: 0, L2 mismatch timeouts: 0, FIFO errors: 0, Resource errors: 0
  Output errors:
    Carrier transitions: 3, Errors: 0, Drops: 0, Collisions: 0, Aged packets: 0, FIFO errors: 0, HS link
CRC errors: 0, MTU errors: 0, Resource errors: 0
    CRC/Align errors                          0                0
    FIFO errors                               0                0
    Total errors                              0                0
    Output packet error count                                  0
```

This is incredibly helpful. You'll find yourself using this a lot in production. (Well, hopefully not too much. I hope for your own sake that your network remains stable!)

Having said that, there are two problems with that command. First, you don't necessarily know when the errors happened. Second, you don't know if the errors are rising. Well, funny I should mention that: I've got two new commands for you that will fix exactly that problem.

Clear and Refresh: Two Helpful Troubleshooting Commands

The first is command is clear: it's `clear`! Ha ha, great stuff. This command can be used any time you want to reset counters. Let's do it now, focusing in on ge-0/0/0:

```
root@Router1> show interfaces ge-0/0/0 extensive | match bytes
  Input  bytes  :          10888872            3059384 bps
  Output bytes  :          10887384            3059392 bps
{snip}

root@Router1> clear interfaces statistics ge-0/0/0

root@Router1> show interfaces ge-0/0/0 extensive | match bytes
  Input  bytes  :                 0            3084744 bps
  Output bytes  :                 0            3085384 bps
{snip}
```

Great! The counters are back to 0, including error counters. Now we can be confident that any further errors we spot will have occurred since we typed that `clear` command. (I'm amazed that I managed to type these commands so quickly that these counters were at 0 – they will have started rising almost immediately!)

What if the counters are rising slowly? Ideally you'd be able to view these counters in real time. Well, you could keep pressing up and return so that our previously-entered command keeps appearing on our CLI. But there's a better way. Use this command to automatically deploy a command every two seconds:

```
root@Router1> show interfaces extensive | match ge-0|error|flap | refresh 2
```

This command will keep typing this command on your behalf, displaying the results on your screen every two seconds, until the end of time. Or, until you press Ctrl-C to cancel it. Or until your power gets cut off because you forgot to pay the bill because you were too busy watching this output—whichever of the three comes first.

Some Additional Filtering Commands

We've talked a lot about `match`. How about `except`? It does the opposite: it shows you all the output, apart from lines that include your specified text. For example, the next command is a quick way to see the status of only the physical gig-ethernet interfaces on the box: by first matching for the text `ge`, and then excluding the text `inet`. Remember, the protocol is configured on the logical interface, not the physical interface:

```
root@Router1> show interfaces terse | match ge | except inet
ge-0/0/0            up    up
ge-0/0/1            up    up
ge-0/0/2            up    up
ge-0/0/3            up    up
ge-0/0/4            up    up
ge-0/0/5            up    up
ge-0/0/6            up    up
ge-0/0/7            up    up
ge-0/0/8            up    up
ge-0/0/9            up    up
```

Now let's take this output and do something extra with it. If you owned a very large Juniper MX router, with dozens of gig-ethernet interfaces, you might be interested to know precisely how many of these interfaces your box currently hosts. Add the count command to the previous command, and the result is a report of how many lines of text the output contains:

```
root@Router1> show interfaces terse | match ge | except inet | count
Count: 10 lines
```

Ten lines means ten interfaces.

Let's introduce yet another new operational command, `show log messages`. You'll learn more about logging in Chapter 4. Junos has an internal hard disk that can store massive amounts of logging information. These files grow to be very large indeed. If you're troubleshooting a live problem, chances are you just want to see the entries at the very bottom of the log file, in other words the most recent log entries. Use the `last` command to view any number of final lines of your output:

```
chris.parker@Production_Box> show log messages | last 10
Dec 18 18:23:48  Production_Box l2cpd[4272]: Initializing PNAC state machines complete
Dec 18 18:23:48  Production_Box l2cpd[4272]: Initialized 802.1X module and state machines
Dec 18 18:23:48  Production_Box l2cpd[4272]: Read ACCESS profile () config
Dec 18 18:23:48  Production_Box ffp[4284]: dynamic-profiles: No change to profiles
Dec 18 18:23:48  Production_Box l2cpd[2348]: Read ACCESS profile () config
Dec 18 18:23:48  Production_Box mib2d[2328]: MIB2D_LIMIT_EXCEEDED: pkg_info_scan: exceeded limit 69:
no more room for new packages, skipped.
Dec 18 18:23:48  Production_Box last message repeated 3 times
Dec 18 18:23:48  Production_Box rpd[2329]: RPD_OSPF_NBRDOWN: OSPF neighbor 10.10.13.3 (realm ospf-v2
ge-0/0/1.0 area 0.0.0.0) state changed from Full to Down due to KillNbr (event reason: local router ID
changed)
Dec 18 18:23:48  Production_Box mgd[2945]: UI_COMMIT_COMPLETED: commit complete
Dec 18 18:23:48  Production_Box mgd[2945]: UI_DBASE_LOGOUT_EVENT: User 'root' exiting configuration
mode
```

Of all the commands that you can use after a pipe, so far we've learned the following: `match`, `except`, `count`, `refresh`, and `last`. Let's round off this section by looking at a few more.

In fact, speaking of more, did you notice earlier when we saw all the options that that there was one called `no-more`? Imagine that you did indeed want to view a full log file, with many hundreds of lines of text. Hitting the space bar to scroll one screen at a time can be pretty tedious. In these situations, use `no-more`. This writes the entire result of your command to the screen in one hit, so you can scroll back up and read it all at your leisure.

Here's a brief overview of some of the rest of the filtering options available to us:

- `find`: This command searches your output for the first occurrence of your search text and shows you the full output from that line onwards.
- `resolve`: If your output contains IP addresses, this command can request that the router try to do a reverse DNS lookup to resolve them to hostnames.
- `save`: You can save any output directly to a named file of your choice! We'll see this in action later on.
- `append`: It's like `save` but adds the output to the end of an existing file.
- `trim`: If you want to make your output slightly thinner horizontally, this command trims however many characters you specify from the left of the output.

Once again, rest assured no one is expecting you to read this one chapter and have all of these combinations memorized. But as you get hands-on with Junos, try typing some of them. Perhaps you might even consider typing a handful of them every time you log on to a box, just for the sake of burning them into muscle memory. Do that and they'll become second-nature in no time at all!

CLI Help

You've learned already that you can type ? at the end of any command to see what options are available to you. If you've used other vendor's OS in the past, you'll know that pretty much everyone offers this on their CLI. However, Junos goes one step further and provides you with a massive chunk of Juniper's online knowledge base, directly inside the router!

You access these help files with the `help topic` and `help reference` commands. The difference between the two is that `help topic` tends to give general information about the operation of a protocol or concept, whereas `help reference` gives guidance on the correct way to actually configure the thing you're searching for. You'll tend to use both to get the answer you're looking for.

There's about 50 top-level topics, including interfaces, routing-options, OSPF, firewall, BGP, and plenty more. Within each of these you'll find an incredible number of options. You can of course use your trusty question mark to type something like `help topic ospf ?` to see all the help options available. Let's look at an example:

```
root@Router1> help topic ospf export
                     Configuring OSPF Routing Policy

   All routing protocols store the routes that they learn in the routing
   table. The routing table uses this collected route information to
   determine the active routes to destinations. The routing table then
   installs the active routes into its forwarding table and also exports them
   back into the routing protocols. It is these exported routes that the
   protocols advertise.

   For each protocol, you control which routes the protocol stores in the
   routing table and which routes the routing table exports into the protocol
   by defining a routing policy for that protocol. For information about
   defining a routing policy, see the JUNOS Policy Framework Configuration
   Guide.

   By default, if a router has multiple OSPF areas, learned routes from other
   areas are automatically installed into area 0 of the routing table.
{snip}
```

The help file in that example goes on to give more background theory, and then shows you how to correctly configure OSPF export policies. By contrast, if you were to type `help reference ospf export`, you would see a shorter help file that focuses in on the actual configuration. For the purpose of saving space we haven't included an example here, so hop into your lab and try it yourself. Look at a few help files, and become familiar with what's available to you. When your box is offline and you have no internet, this feature might well save the day!

What if you know that there's something you want to show or configure, but you can't quite remember what the correct command is? In this case, `help apropos` is your friend: it shows you every single command that uses the word. Let's say

you're starting to become familiar with SFP modules, to allow your Juniper box to accept optical cables. The command to look at the stats is on the tip of your tongue, but you can't quite remember it. Type this to see every command that features the word optics, including help files:

```
root@Router1> help apropos optics
clear interfaces transport optics
    Clear interface transport optics information
clear interfaces transport optics interval
    Clear interface transport optics interval information
show interfaces transport optics
    Show interface transport optics information
show interfaces transport optics interval
    Show interface transport optics interval information
show interfaces diagnostics optics
    Show interface optics-diagnostics information
show chassis fabric optics
    Show info of fabric optical ports
request chassis optics
    Change linecard optics status
request chassis optics fpc-slot <fpc-slot>
    Slot number of FPC that houses optics
request chassis optics reactivate
    Reactivate linecard optics
request chassis sib optics
    Change SIB optics status
request chassis sib optics optics-slot <optics-slot>
    Optics slot number
request chassis sib optics enable
    Enable the optics
request chassis sib optics disable
    Disable the optics
help reference interfaces optics-options
    Optics configuration for 10-Gigabit Ethernet DWDM PIC
```

One final help option is the extremely useful `help syslog` command. This one is a bit tricky to explain, because we've not yet taught you about logging: that will come in Chapter 4. For now, let's just say that it's possible to set up your logging in such a way that many of these logs contain a thing called a syslog message. This is a short piece of text in capital letters that uniquely identifies what the log actually is, and you can use this syslog message to get more information about what the log actually means. For example, next I'll show you a small section of a much larger text file. Take a look at the log that has `LACPD_TIMEOUT` in it:

```
chris.parker@Production_Box> show log messages
{snip}
vApr 20 17:25:32  Production_Box jdhcpd: shmlog: shared log header is NULL
Apr 20 18:01:08  Production_Box lacpd[1998]: LACPD_TIMEOUT: xe-1/0/14: lacp current while timer
expired current Receive State: CURRENT
Apr 20 18:01:08  Production_Box /kernel: lag_bundlestate_ifd_change: bundle ae16: bundle IFD minimum
bandwidth or minimum links not met, Bandwidth (Current : Required) 0 : 10000000000 Number of links
(Current : Required) 0 : 1
{snip}
```

It looks like interface `xe-1/0/14` has an LACP problem. Except, what is LACP? Perhaps that log message might mean something to you if you're a seasoned engineer, but if you're a newcomer it could be a bit confusing. Luckily, you can use this command to get Junos to tell you the answer:

```
chris.parker@Production_Box> help syslog LACPD_TIMEOUT
Name:        LACPD_TIMEOUT
Message:      <error-message>: lacp current while timer expired current
            Receive State: CURRENT
Help:        lacp timer expired
Description:  The Link Aggregation Control Protocol process (lacpd)
            experienced timeout hence Receive State will change from Current
            to Expired/Defaulted.
Type:        Event: This message reports an event, not an error
Severity:    notice
Facility:    LOG_DAEMON
```

Aah! So it turns out that LACP stands for Link Aggregation Control Protocol! Sounds like interface xe-1/0/14 is part of some larger bundle of aggregated Ethernet interfaces. You can see how helpful this command can be when you're looking at confusing log messages! Be aware though that not every log message has one of these syslog tags – and not every single syslog tag has a help message associated with it. But a huge majority of them do, so it's well worth making it your first port of call in times of troubleshooting.

Before we leave the online help files, there are two extras I want to show you. They're not essential, but they are very cute.

First, let me tell you about a command many engineers don't know about: `help tip cli`. Type this to get a random tip about the CLI:

```
root@Router1> help tip cli
JUNOS tip:
Use 'request message user <foo>' to send a text message to one
user. Use 'request message all' to send to all current users.
```

See: even Juniper encourages you to play tricks on your colleagues. (This is a joke. EXTREMELY a joke. Juniper in no way encourages you to play tricks on your colleagues.)

(Having said that, I absolutely do encourage it. It's why they call me *Chris "Best Practices" Parker*.)

If you like, type a random number after that command. This number gets stored so you can come back to it later.

Finally, here is a hidden command that is of no practical value at all, but will brighten up your day: `show version and haiku`. Yes, as you check what model of box you're on, you can also request that Junos gives you a poem in the famous Japanese style. You can't tab-complete this command; it's a secret command, shared only among engineers with a good heart:

```
root@Router1> show version and haiku
Hostname: Router1
Model: vmx
Junos: 14.1R1.10
{snip}

        Haikus are magic
        Springing from your CLI
        Wasting half your day
```

If you want to get fired from your job, why not try sitting at your desk all day issuing this command to see how many haikus Junos stores?

We've covered operational mode fairly extensively here, though there's still plenty of commands left. For example, in Chapter 4 you'll learn some details about ping and traceroute that you might not get on a Windows or Mac box. You'll also learn about the `monitor` command, which allows you to view interfaces stats and the contents of files in real time. This lets you view new log files and new debugging information, without putting too much strain on the CPU.

For now, let's move on to configuration mode - but before we begin, I ask a personal favor of you. Whatever your experience is of configuring the devices of other vendors, whatever knowledge you have of terminology and process, please put it to one side. Chances are that Junos does it slightly differently and by the end of this chapter I hope you'll agree with me that Junos does it far, far better.

Viewing the Configuration

Before we actually enter configuration mode, it makes sense to look at one more operational command: `show configuration`. Or `sh config` for short!

I've set up my vMX to have a bare-bones config, to make it easier for you to read. If you've got a real box in front of you, your config might be longer. In any case, let's look at my configuration in full, and analyze what we see.

As you look at the output, try reading it all, word for word, out loud if need be. You'll immediately notice that the configuration looks almost like a programming language. After you've read it in full, we'll break down what's happening:

```
root@Router1> show configuration
## Last commit: 2019-12-18 20:41:29 UTC by root
version 14.1R1.10;
system {
    host-name Router1;
    root-authentication {
        encrypted-password $1$NvYkG5me$TktHnF7/KfTXIYIV4jFxa/; ## SECRET-DATA
    }
    syslog {
        user * {
            any emergency;
```

```
        }
        file messages {
            any notice;
            authorization info;
        }
        file interactive-commands {
            interactive-commands any;
        }
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 10.10.12.1/24;
            }
        }
    }
    ge-0/0/1 {
        unit 0 {
            family inet {
                address 10.10.13.1/24;
            }
        }
    }
}
routing-options {
    autonomous-system 64512;
}
protocols {
    ospf {
        area 0.0.0.0 {
            interface ge-0/0/0.0;
            interface ge-0/0/1.0;
        }
    }
}
firewall {
    family inet {
        filter EXAMPLE_FIREWALL_FILTER {
            term DENY_SSH {
                from {
                    protocol tcp;
                    port ssh;
                }
                then {
                    discard;
                }
            }
            term ACCEPT_ALL_ELSE {
                then accept;
            }
        }
    }
}
```

Look at the far left-hand side of the output. We see the words `system`, `interfaces`, `routing-options`, `protocols`, and `firewall`. On closer inspection, you can notice that everything indents outwards from these keywords.

This is no accident: once again, Junos configurations are extremely *hierarchical*, which means that you can not only predict where you'll find certain configuration options with dazzling accuracy, it also means you can guess your way through a great deal of the configuration you'll need to deploy. There are not many vendors that can offer that!

For example, look at the `interfaces` hierarchy. Notice that indented underneath the `interfaces` keyword we have `ge-0/0/0` and `ge-0/0/1`. Then, underneath each of these interfaces we see commands that are further indented. As you might guess, the text underneath each *stanza* applies only there, to that particular interface.

Keep an eye out for the opening curly bracket `{` at the start of the line. This is an indication that the following line is one step down in the hierarchy. When you see a semicolon, it means that particular command is complete. When you finally see a closing curly bracket `}`, this indicates that there's nothing left at that level of the hierarchy. That's why there's some single lines that have only a closing curly bracket. They allow your eyes to quickly identify where something is within the hierarchy.

For example, take a look at the `firewall` stanza. You'll see I made a fake firewall filter, which I named `EXAMPLE_FIREWALL_FILTER`. I am nothing if not original. Can you see that there are two terms in this filter? And can you see that the first term has a `from` and `then` clause? You'll see that the curly brackets are exactly aligned to where everything is within the hierarchy. Scale this out to fifty firewall filters, each with twenty terms, and you'll really appreciate how readable this all is, thanks to the indentations quickly showing you where the different terms start and end.

Let's talk about a few of the other top-level stanzas we saw in the example above, and let's introduce a few new ones, too:

- `system`: In this stanza you'll find configuration relating to the device itself. Users, passwords, name servers, logging, and configuration archival - these are just a small selection of the features you'll configure here.
- `interfaces`: This one does what it says on the tin! However, it's important to note that you only configure the interface itself here. If you want to enable a protocol on an interface, you do that in the `protocols` hierarchy.
- `protocols`: Every single protocol you'd want to turn on will be found here. BGP, OSPF, LDP, IS-IS, RSTP, PIM, RIP, and many more. If some of those don't mean anything to you yet, don't worry – they will in time!
- `routing-options`: This is where you'll configure static routes, set your router ID, and some create particularly advanced route manipulation. In Chapter 5:

Routing Fundamentals and Chapter 6: Routing Policies, you'll definitely be making changes in here.

- `policy-options`: Later in this book you'll come to learn that routing protocols – the languages routers that can talk to learn where IP addresses are - have a default behavior, which you can override with routing policies. This is the section where you'll configure these policies, specifying exactly which addresses you want to receive and advertise.
- `routing-instances`: In the same way that we can virtualize a physical computer into multiple virtual computers, we can even do something akin to virtualizing our router into many virtual routers! There are a few ways of doing this, and they're all done in this stanza.

Newcomers often look at this indented configuration and wonder how on earth you configure the box. Do you have to type the configuration in the same way, with curly brackets and tabbed spaces in the correct place? No, not at all. Let's look at a section of the configuration as an example. Notice that in this show command I'm not just typing `show configuration`, but `show configuration interfaces ge-0/0/0`, to drill down to just the specific bit that I'm interested in viewing:

```
root@Router1> show configuration interfaces ge-0/0/0
unit 0 {
    family inet {
        address 10.10.12.1/24;
    }
}
```

TIP In this example, also notice that the first two levels of the hierarchy - interfaces and ge-0/0/0 aren't displayed. The configuration starts at the level I requested in my command.

TIP2 In this and many other Junos books, the bold part of the first line of a CLI snippet is what you physically type, and what follows is what Junos displays to your boldface query.

The configuration is displayed hierarchically, but to actually configure the box you must enter configuration mode (in a moment I'll show you how to do this) and simply type the command below. Notice that there are no curly brackets: it's all just typed as one line. Follow it along and you'll notice that the output above, and the command below, line up precisely:

```
[edit]
root@Router1# set interfaces ge-0/0/0 unit 0 family inet address 10.10.12.1/24
```

So, it's typed as one single line but displayed in that indented, hierarchical format. This gives us the advantage of making configurations easy to read and understand, particularly when we're dealing with complicated routing policies and firewall filters, but without burdening us with typing that's full of curly brackets!

If you don't want to view the configuration in hierarchical format, and instead want to view it as if it were a string of `set` commands, you can do that too. Type `show configuration | display set`, or add `display set` to any piece of configuration you're interested in viewing:

```
root@Router1> show configuration interfaces | display set
set interfaces ge-0/0/0 unit 0 family inet address 10.10.12.1/24
set interfaces ge-0/0/1 unit 0 family inet address 10.10.13.1/24
```

At this point, if you're coming from another vendor's OS, you might look at these `set` commands and think that they involve more typing than your vendor of choice. And to a certain extent, that's true. The advantage, though, is that you'll often find the full command more human-readable. It also gives you the advantage of the predictable hierarchy. And in addition to that, there's actually a way to speed up these commands. In a moment I'll show you how.

In my experience, almost every new student I've ever taught learns the Junos `display set` command, and declares that they're just going to always use that to view the configuration. They'll say they can see more on the screen at once, and it's more familiar to them if they've come from other vendors. That's absolutely fine! There's no right or wrong way to use these commands.

However, as you get further into Junos and you start configuring these complicated routing policies and firewall filters, or as you configure interfaces with many features beyond mere IP addresses, I promise you that you'll come back to the hierarchical format more and more. Long commands can be tricky to read in `display set` format. By contrast, the hierarchy spaces things out, and makes it logical to read. As you become more familiar with the philosophy of Junos, you'll be very comfortable with the advantages of both views.

Finding Everything You Want in a Configuration File

The behavior of the hierarchy is very consistent. For example, in future chapters you'll learn about routing protocols like OSPF. If you want to turn OSPF on for an interface, you always do it under the `set protocols ospf` hierarchy, not under the `set interfaces` hierarchy. This is great. It means that all the OSPF settings are in one place, and you don't need to be confused about where you'll find certain configuration elements.

You might wonder though: if the full configuration for an interface isn't entirely under the `interfaces` hierarchy, what do I do if I want to view all the configuration that relates to a certain interface? For example, what if I want to see the IPs configured on an interface and also the protocols configured on an interface? Do I have to type multiple `show configuration` commands to see it all? Thankfully not. There's a much easier way.

First, type `show configuration | display set` to show the entire configuration in set format. Now, add this to the end of the command:

```
root@Router1> show configuration | display set | match ge-0/0/0
set interfaces ge-0/0/0 unit 0 family inet address 10.10.12.1/24
set protocols ospf area 0.0.0.0 interface ge-0/0/0.0
```

This command provides the best of both worlds: it means we can keep 100% of our protocol-related configuration strictly under the `protocol` hierarchy, but also view all mentions of the interface throughout the entire config.

There are more `display` commands you can use than just `display set`. For example, imagine that your configuration has dozens of firewall filters. Typing `show configuration` would give you a configuration that is perhaps much longer than you'd hope to see. In this situation, it might be useful to actually hide firewall filters when you view a configuration, to keep things readable. In configuration mode (honestly, we're very close to learning how to get into it!) one command is all you need:

```
[edit]
root@Router1# set firewall apply-flags omit
```

Now when we view this part of the configuration it will actually be hidden. I've snipped the rest of the output, to show you just the effect of this command:

```
root@Router1> show configuration
{snip}
firewall { /* OMITTED */ };
```

If you want to see the entire configuration, all you have to do is type `show configuration | display omit` to bring it back again. Alternatively, doing a `display set` will always show you the full configuration, even if something has been tagged as omitted.

The `apply-flags omit` command is actually hidden. You won't be able to tab-complete it. Some commands are hidden to prevent casual users from using them, and to stop expert users from accidentally typing them. You have to actively choose to use this command!

There's also an option called `display inheritance`. You'll learn about this in the "Configuration Groups" section of Chapter 3. For now, just know that typing `display inheritance` allows you to view the full compiled configuration, which may include pieces that are defined once and then referred to multiple times in the configuration for consistency.

One final thing, which is very cool to know: behind the scenes, the configuration is actually stored as XML. In the future, as you progress into Junos automation, you might well find yourself referring directly to this XML. Try going into your lab and typing `show configuration | display xml`. Pretty neat, eh?

Configuring Your Device: Active Configs Versus Candidate Configs

Finally, finally! We're going to actually configure our device.

To go into configuration mode, type `configure`. You can also type `edit`, which is why you hear some engineers call configuration mode *edit mode*. Both commands are fine but I prefer to say *configure*. Why? The word edit has an addition meaning when in configuration mode, as we'll soon see, and it's nice to have a clean separation of meaning between the two terms.

When you do enter configuration mode, you'll see output like this:

```
root@Router1> configure
Entering configuration mode

[edit]
root@Router1#
```

The `[edit]` at the top tells us that we're ready to start typing full set commands. Notice as well that the `>` has changed to a `#`, to signify that we're in configuration mode, not operational mode.

The current, live, running configuration on a Junos device has a special name: it's called the active configuration.

If you've used the devices of other vendors, you're possibly familiar with the concept of there actually being two configurations at any one time. That's because on those boxes you'll often find that as soon as you type a configuration command, that command goes live straight away – but it isn't saved. If you were to reboot the box, you'd lose all your changes, until you manually save your work. In other words, a lot of vendors have a distinction between the current running configuration and the actual saved configuration. The running configuration can be vastly different to the saved configuration, and the two become ever more different with every command you type.

Junos is different. Very different. To see how, let's try changing the hostname of my box from Router1 to Router_1:

```
[edit]
root@Router1# set system host-name Router_1

[edit]
root@Router1#
```

Hmm. The hostname stayed the same. What gives? Make sure you're sitting down: the following explanation is going to change the way you do networking forever.

Instead of making our configuration go live straight away, Junos puts this change into a thing called the candidate configuration. Every single configuration change

you type goes into it, and they stay there until you choose to deploy them (all at once) to your live environment. You have the option to make multiple changes before deployment, and crucially the chance to review your changes, and correct any fatal mistakes before they go live!

Now that we've added the configuration to change our hostname, let's make an extra change and tell our box what name-server to use to resolve domain names. Then let's compare our candidate configuration to the active configuration:

```
root@Router1# set system name-server 8.8.8.8

[edit]
root@Router1# show | compare
[edit system]
-  host-name Router1;
+  host-name Router_1;
+  name-server {
+      8.8.8.8;
+  }

[edit]
root@Router1#
```

Did you notice that I didn't type `show configuration | compare`, but just `show | compare`? When you're in configuration mode the `show` command acts a little differently than it acts in operational mode. Typing `show` on its own simply shows you the entire candidate configuration, combining existing configuration with your new changes. By contrast, using `show | compare` allows you to zoom in on just the pieces of the configuration that are changing.

To see how `show` acts differently in configuration mode, try typing `show interfaces` in configuration mode. Rather than showing you interface stats (as you'd expect to see from this command in operational mode), you'll instead see the full configuration of the `interfaces` stanza in the candidate configuration.

I'm sure `show | compare` will soon become your new best friend. As you can see, every line we're adding has a plus sign by it. Equally, if any lines are going to be removed from our active configuration, you'll see a minus sign by them.

Now bear in mind that at the moment, none of these changes have been saved. None of these changes have gone live. They all still exist only in the candidate configuration. In a few pages time, you'll learn how to save that configuration. For now, let's learn how to correct mistakes.

What if I realize that I typed my new host-name incorrectly? There's two ways to fix it. In this particular situation, the first option is to simply type the command again with the correct host-name. Sometimes, re-typing the correct command is enough to overwrite what you previously typed. However, be careful: some commands involve actively deleting what you previously typed. We'll see examples of that in a moment.

For now, let's correct the host-name:

```
[edit]
root@Router1# set system host-name Router_Chris

[edit]
root@Router1# show | compare
[edit system]
-  host-name Router1;
+  host-name Router_Chris;
+  name-server {
+      8.8.8.8;
+  }

[edit]
root@Router1#
```

The second way to fix a mistake is to completely scrap your changes, and start again. Sounds complicated, right? Nope: it's one command. Try typing `rollback` and see what happens:

```
[edit]
root@Router1# rollback
load complete

[edit]
root@Router1# show | compare

[edit]
root@Router1#
```

And with that, all of our proposed changes are gone. The candidate configuration is now exactly the same as the active configuration. It's as if you never typed anything. From here you can either start again, or just log off and go home. Ask your line manager for the correct course of action here.

Are you starting to fall in love with Junos? I thought you might be.

While we're talking about it, allow me to let you in on a nice little time-saver. See, by default, *you can't type any operational commands in configuration mode*. The two modes are different. Operational mode is for operational commands. Configuration mode is for configuration commands. But take my word for it: it's a real bore having to go in-and-out of configuration mode just to type `show interfaces` or `ping`.

Luckily, there's a way around it. Pretty much every operational command – `show`, `ping`, `telnet`, and so on – can actually all be run in configuration mode, if you add the word `run` to the command:

```
[edit]
root@Router1# ping
              ^
unknown command.
```

```
root@Router1# run ping 10.10.13.3
PING 10.10.13.3 (10.10.13.3): 56 data bytes
64 bytes from 10.10.13.3: icmp_seq=0 ttl=64 time=2.531 ms
64 bytes from 10.10.13.3: icmp_seq=1 ttl=64 time=2.337 ms
^C
--- 10.10.13.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.337/2.434/2.531/0.097 ms

[edit]
root@Router1#
```

Interestingly, a great deal of documentation, including a lot of Juniper's official KB articles, show you example CLI output that features `show` commands entered from configuration mode. Indeed, some engineers never leave configuration mode at all, opting to always just type `run` before their commands.

Be warned though: this is the very opposite of a best practice. When you're in configuration mode all the time, it's very easy to type something you didn't mean to type. Get into the habit of going into configuration mode when, and only when, you need to. You're about to learn that Junos offers you a number of helpful ways to prevent yourself from accidentally deploying incorrect configuration. Nevertheless, don't tempt fate. Enter configuration mode only when you have a requirement to be there. When you're done, type `exit` to leave configuration mode:

```
[edit]
root@Router1# exit
Exiting configuration mode

root@Router1>
```

Configure Exclusive and Configure Private

One thing that's important to understand about the candidate configuration is that by default there is one single candidate configuration that all users share. If two users enter configuration mode at the same time, and enter separate configuration commands, both sets of changes all go into the one shared candidate configuration. As a result, if one user saves their changes – and later on we'll find out how we do this – they'll actually be saving both their own changes and the changes made by their colleagues. This may or not be want you want to do.

Is there really a danger of two people going into configuration mode at the same time, and saving each other's work by mistake? Yes, although as the second user enters into configuration mode, they'll be prompted that another user is already logged in:

```
root@Router1t> configure
Entering configuration mode
Users currently editing the configuration:
```

```
  chris.parker terminal pts/1 (pid 26396) on since 2020-04-18 23:01:39 UTC, idle 00:00:46
      [edit]

[edit]
root@Router1#
```

But what if that's not enough? What if you want to *guarantee* that someone else won't come in and make additional changes that you'll commit by mistake? In this case, you have two options. First, use `configure exclusive`:

```
root@Router1> configure exclusive
warning: uncommitted changes will be discarded on exit
Entering configuration mode

[edit]
root@Router1#
```

When you do this, your colleagues won't even be able to enter configuration mode until you're done making your changes. This gives you complete security and peace of mind that you're saving your changes, and only your changes.

Alternatively, perhaps you might be happy for others to make configuration changes during your own change window but you still don't want to accidentally commit something that someone else is halfway through typing. In that case, type `configure private`. This gives you your own candidate configuration to work on. Other users will also have to type `configure private` while you're doing this, to get their own private candidate config. This method even protects against two users configuring the same thing: in private mode Junos prompts the second user's attempt to commit a change to a stanza that the first user has worked on.

As you can see, even the act of going into configuration mode is much more involved than on the average networking device! But you can also see that these extra options give you far more control. Once again, a few extra commands give you an exponential increase in security.

Modifying Configurations: An Introduction to the Set Command

Now that we're finally in configuration mode, let's learn about how we actually make changes. Remember that configuration commands are deployed using the `set` keyword when we configure an interface and give it an IP address:

```
[edit]
root@Router1# set interfaces ge-0/0/1 unit 0 family inet address 10.10.13.1/24
```

If you follow this command from start to end, you'll see a predictable hierarchy. First we go into `interfaces ge-0/0/1`, then we mention that we're specifically editing the logical sub-interface `unit 0`. We want to put an IP address on it – but what kind of IP address? An address in the `inet` family. If we wanted to add an IPv6 address, the command would be like this:

```
[edit]
root@Router1# set interfaces ge-0/0/1 unit 0 family inet6 address 2001:db8:0:13::1/64
```

Once saved, the interface configuration now looks like this:

```
root@Router1> show configuration interfaces ge-0/0/1
unit 0 {
    family inet {
        address 10.10.13.1/24;
    }
    family inet6 {
        address 2001:db8:0:13::1/64;
    }
}

root@Router1>

root@Router1> show configuration interfaces ge-0/0/1 | display set
set interfaces ge-0/0/1 unit 0 family inet address 10.10.13.1/24
set interfaces ge-0/0/1 unit 0 family inet6 address 2001:db8:0:13::1/64
```

Cool! Notice how the two families are in the same level of indentation in the hierarchy. Again, when you truly understand this hierarchy, configurations are so easy to read. But don't worry if it doesn't click right away. After you've read a few configurations of your own, I promise that it will click very quickly.

Configuration Mode CLI Navigation

Let's imagine that I wanted to configure a brand new interface, ge-0/0/2, and give it both an IP address and an IPv6 address. From what we've learned so far, I could do it like this:

```
[edit]
root@Router1# set interfaces ge-0/0/2 unit 0 family inet address 10.10.14.1/24

[edit]
root@Router1# set interfaces ge-0/0/2 unit 0 family inet6 address 2001:db8:0:14::1/64

[edit]
root@Router1#
```

However, you might reasonably conclude that we had to type a lot to get those two addresses on the interface. Is there a quicker way?

One way would be to press the Up key after the first command, press Ctrl-W three times to delete the text `inet address 10.10.14.1/24`, and then type `inet6 address 2001:db8:0:14::1/64`, instead. This would work, and it would certainly be quicker than typing the whole thing again!

But there's an even cleaner way. Instead of `set`, try typing `edit interfaces ge-0/0/2 unit 0`, and see what happens to your command prompt:

```
[edit]
root@Router1# edit interfaces ge-0/0/2 unit 0

[edit interfaces ge-0/0/2 unit 0]
root@Router1#
```

Notice that the word `[edit]` has changed to `[edit interfaces ge-0/0/2 unit 0]`. You've just drilled down in the hierarchy! Now that you're here, any `set` commands you type are applied only at this level of the hierarchy – meaning you don't need to type `set interfaces ge-0/0/2 unit 0` over and over.

At this point the use case for this `edit` command may not immediately be clear - after all, we're only adding two lines of configuration. But scale it out. Imagine a firewall filter with fifty terms. The ability to type something like `edit firewall family inet filter EXAMPLE_FIREWALL_RULE` just once to go into that piece of the hierarchy is very helpful indeed!

Let's see what this looks like in full:

```
[edit]
root@Router1# edit interfaces ge-0/0/2 unit 0

[edit interfaces ge-0/0/2 unit 0]
root@Router1# set family inet address 10.10.14.1/24

[edit interfaces ge-0/0/2 unit 0]
root@Router1# set family inet6 address 2001:db8:0:14::1/64

[edit interfaces ge-0/0/2 unit 0]
root@Router1# top

[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }

[edit]
root@Router1#
```

Did you notice that I typed the command `top`? This command quickly takes me back to the very top of the hierarchy. You can even combine `top` with `show` or `edit`, all on one line. For example, imagine I was in the `set protocols bgp` hierarchy, and I wanted to view the OSPF configuration – but still stay where I was within the hierarchy. To do that, I simply type the commands in the output below:

```
[edit protocols bgp]
root@Router1# show

[edit protocols bgp]
root@Router1# top show protocols ospf
area 0.0.0.0 {
    interface ge-0/0/0.0;
    interface ge-0/0/1.0;
}

[edit protocols bgp]
root@Router1#
```

Notice that the first `show` command gives me nothing, because I have no BGP configuration either in my active or candidate configurations. However, the second command gives me something valuable – and crucially, it keeps me where I am in the hierarchy.

As an alternative to `top`, I could have typed `up` to only go up one single level of the hierarchy:

```
[edit protocols bgp]
root@Router1# up

[edit protocols]
root@Router1#
```

You can even type something like `up 3` if you want to move up three levels of the hierarchy. As you become a master of firewall filters and routing policy, you'll find that options like this are real time savers. Soon you'll be able to navigate around configuration mode at lightning speed!

At this stage, though, let's be clear on something: reading all of this in a book is one thing, but I promise you that the only way it will truly stick in your mind is if you type all of these commands yourself. If you're not doing so already, fire up a box in a lab, and type all of these commands yourself. See what happens. Press `?` now, and again as you go, to see what other options are available to you. Get hands-on. There's a lot to remember, but if you actually do it, you'll learn it very quickly.

Hey, remember that `help apropos` command from earlier? It even works in configuration mode – and it's smart enough to know where you are in the hierarchy! What if you were about to configure some BGP, but you couldn't remember how to set a neighbor to talk BGP with? Junos has you covered:

```
[edit protocols bgp]
root@Router1# help apropos neighbor
set path-selection always-compare-med
    Always compare MED values, regardless of neighbor AS
set group <group_name> as-override
    Replace neighbor AS number with our AS number
set group <group_name> neighbor
    Configure a neighbor
set group <group_name> neighbor <address> as-override
```

```
    Replace neighbor AS number with our AS number

[edit protocols bgp]
root@Router1#
```

In fact, pretty much every command you've learned so far is hierarchy-aware. Try going into this stanza, and then typing `show`:

```
[edit]
root@Router1# edit interfaces ge-0/0/1 unit 0

[edit interfaces ge-0/0/1 unit 0]
root@Router1# show
family inet {
    address 10.10.13.1/24;
}
family inet6 {
    address 2001:db8:0:13::1/64;
}

[edit interfaces ge-0/0/1 unit 0]
root@Router1#
```

Great! This is a nice way of seeing just the piece of the candidate configuration that you want to see in that moment. You can still type `display set` in this mode, too:

```
[edit interfaces ge-0/0/1 unit 0]
root@Router1# show | display set
set interfaces ge-0/0/1 unit 0 family inet address 10.10.13.1/24
set interfaces ge-0/0/1 unit 0 family inet6 address 2001:db8:0:13::1/64
```

Or, perhaps you just want to see the `set` commands relative to where you actually are in the hierarchy? No worries:

```
[edit interfaces ge-0/0/1 unit 0]
root@Router1# show | display set relative
set family inet address 10.10.13.1/24
set family inet6 address 2001:db8:0:13::1/64

[edit interfaces ge-0/0/1 unit 0]
root@Router1#
```

Try all of these out for yourself. Some you'll just use occasionally, others you'll find yourself using every single day. Just be careful to go back to the `top` before you actually commit anything. If you do a `show | compare` deep in the hierarchy, you'll only see the changes you've made at that part of the hierarchy – which means you might unwittingly save config you didn't mean to save.

Take a look at the output below. I give a new interface a description, then I move into the `system` hierarchy and change the hostname. When I check my changes, only the host-name change is displayed. But then when I commit my configuration, it's clear that the interface change was saved too! (We've not covered the "commit" command yet, but in a moment we'll talk about in great detail):

```
[edit]
root@Router1# set interfaces ge-0/0/3 unit 0 description Interface reserved for great justice

[edit]
root@Router1# edit system

[edit system]
root@Router1# set host-name NewRouter

[edit system]
root@Router1# show | compare
[edit system]
+ host-name NewRouter;

[edit system]
root@Router1# commit
commit complete

[edit system]
root@NewRouter# top

[edit]
root@NewRouter# run show configuration interfaces ge-0/0/3 | display set
set interfaces ge-0/0/3 unit 0 description Interface reserved for great justice

[edit]
root@NewRouter#
```

Modifying Configurations: Delete

Let's imagine that we've set a new IP address on the previously empty interface ge-0/0/2. At this stage, we haven't saved our work:

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
```

What if it turns out I've typed the IP address wrong and I want to correct it? Earlier we saw that we simply had to type a new hostname to replace the incorrect one. The reason we were able to simply type the command again with the right information is because you can only have one hostname on a box.

By contrast, it's perfectly possible to have more than one IP address on an interface. Indeed, in IPv6 we do frequently have multiple addresses. For that reason, simply typing a new IP address doesn't have the effect you think it might have. It doesn't replace the old address; it keeps it there and adds a second address.

Imagine that you made a typing mistake. You actually meant to use 10.10.41.1/24 on ge-0/0/2's unit 0. Let's correct it by issuing this command, with the correct IP address:

```
[edit]
root@Router1# set interfaces ge-0/0/2 unit 0 family inet address 10.10.41.1/24
```

Now let's check our candidate configuration and see whether this really worked:

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+               address 10.10.41.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }

[edit]
root@Router1#
```

Look at that. Our second command didn't over-write the first command at all. Instead, it's added both IPs to the interface! In Chapter 3 you'll learn why this is possible, and you'll be introduced to the concept of *primary* and *preferred* IP addresses. For now though, let's use the `delete` keyword to remove the incorrect line:

```
[edit]
root@Router1# delete interfaces ge-0/0/2 unit 0 family inet address 10.10.14.1/24
```

Did it work?

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.41.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
```

```
+       }
+   }

[edit]
root@Router1#
```

Perfect. You can use this `delete` command to undo any unwanted `set` commands, even when you're deep down in the hierarchy.

Be careful though. You might think the `delete` command is simple and self-explanatory but there's a bit more to this command than meets the eye. You see, the hierarchical nature of Junos means that using delete on a full line of configuration doesn't necessarily delete that entire line. Instead, it often just deletes the final part of the line. To see what we mean, let's imagine we also wanted to delete the two remaining addresses. We type this:

```
[edit]
root@Router1# delete interfaces ge-0/0/2 unit 0 family inet address 10.10.41.1/24

[edit]
root@Router1# delete interfaces ge-0/0/2 unit 0 family inet6 address 2001:db8:0:14::1/64

[edit]
root@Router1#
```

Now that we've deleted the two addresses that we previously added, we've successfully removed all the configuration we added to the candidate... right? We should now expect to see a completely empty candidate configuration... right?

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet;
+           family inet6;
+       }
+   }

[edit]
root@Router1#
```

In fact, no. Only the final piece of the hierarchy, the address stanza, has been deleted. This is actually a good thing. Imagine if we had an interface with a variety of other settings on it: VLAN tags, speed/duplex settings, perhaps a user-defined description, and so on. In this scenario, it would be disastrous if typing `delete interfaces ge-0/0/2 unit 0 family inet6 address 2001:db8:0:14::1/64` deleted the entire interface!

To completely delete the interface, we'd need to type `delete interfaces ge-0/0/2`. This deletes interfaces ge-0/0/2, and everything underneath it in the hierarchy.

In this example, the inet config that remains in the candidate configuration is almost certainly benign. However, the remaining inet6 config will still mean that the box has a link-local IPv6 address, and talks IPv6 on the link. This may or may not be your intended behavior.

As you read the later chapters in this book and learn how to configure routing protocols and firewall filters in more detail, think to yourself how you might go about safely deleting certain aspects of the configuration – and whether you're really deleting everything that you want to delete, or whether you're just deleting a tiny piece of the full configuration that you actually want to remove instead.

There's yet another advantage to this hierarchical configuration – you can use wildcards to delete multiple stanzas at once. In your lab, try typing `wildcard delete interfaces ge-0/0/*`. Junos will prompt you to make sure you're not drunk, after which you will have successfully deleted all the gig-ethernet interfaces on that particular line-card, all with one command. You'll notice that I chose `no` in the output below, because I didn't want to actually delete them: I just did it as an example. The `(no)` in brackets indicates that if you were to just hit return, `no` would be the default answer:

```
[edit]
root@Router1# wildcard delete interfaces ge-0/0/*
  matched: ge-0/0/0
  matched: ge-0/0/1
  matched: ge-0/0/2
Delete 3 objects? [yes,no] (no) no


[edit]
root@Router1#
```

This ability to easily delete large sections of config at once means that you can just as easily type something like `delete protocols` to delete the entirety of the `protocols` hierarchy. In your lab, this is a blessing. You can quickly and easily remove large options of configuration in one go. On the other hand, in a production environment this ease with which you can delete all protocols might seem like a curse. I once heard of an engineer many years ago who accidentally deleted all the BGP configs on a production box made by a vendor whose commands go live immediately. Suffice it to say, it wasn't the happiest of days.

Luckily though, no change goes live straight away in Junos. Make sure to regularly use `show | compare` before any and every commit, and you'll always see the error of your ways, allowing you cleanly `rollback` to your active configuration. Even if you did delete every single interface on your Junos device, you could get rid of your changes right away and no one would be any the wiser.

Modifying Configurations: Rename, Replace Pattern, and Copy

Based on what you've learned so far, you might imagine that the correct way to change the IP address on an interface is to add the new address and then delete the old one. And that certainly works. Luckily though, there's an even easier way. Let me introduce you to the `rename` function.

Earlier on I configured an IP address on interface ge-0/0/2.0. I set it to 10.10.14.1/24. I then deleted it, and added a new address, 10.10.41.1/24. But what are the chances! It turns out that I was correct the first time! Let's see how we can use rename to quickly change it back:

```
[edit interfaces ge-0/0/2 unit 0]
root@Router1# rename family inet address 10.10.41.1/24 to address 10.10.14.1/24

[edit interfaces ge-0/0/2 unit 0]
root@Router1# top show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
```

The result speaks for itself – in situations where it's possible to have more than one user-defined attribute, use `rename` to easily change whatever is there.

What about if you want to change every single instance of some text throughout a configuration? For example, what if you notice that your ge-0/0/2 interface is broken, and you want to fix the issue by moving both the cable and the configuration to your ge-0/0/3 interface?

Unplugging the cable is easy. But what about changing the configuration? Well, you now know that you could quickly and easily edit the configuration on the interface itself by using the `rename` keyword. But sometimes it won't be as easy as that. In the real world, you'll also need to edit every other mention of the interface throughout the configuration – `protocols ospf`, `protocols lldp`, `routing-instances` – there's a lot to check. What's the best way to do it?

One option would be to `show configuration | display set | match ge-0/0/2`. You could paste the dozen or so lines of result into a text editor and then paste them again so you have two copies. For the first bunch of commands, you could change the word `set` on each line to `delete`. On the second lot you could change `ge-0/0/2` to `ge-0/0/3`. You could then paste all of that back in, all in one go. Wow! What a time saver! What great automation!

Except, don't do that because – and I'm fully aware that this is my catch-phrase now – there's a better way. Using the command `replace pattern`, you can change every single entry for ge-0/0/2 to ge-0/0/3.

To highlight the impact this has, I've gone ahead and configured interface ge-0/0/2 in a few other places on my vMX. Some of these new lines of configuration will already make sense to you based on what you've learned so far, whereas other parts hint at things to come later on in your studies. In any case, as always, I've not saved my work yet. Now that I've added a load of config for ge-0/0/2, this is my new candidate configuration:

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
[edit protocols]
+   mpls {
+       interface ge-0/0/2.0;
+   }
[edit protocols ospf area 0.0.0.0]
      interface ge-0/0/1.0 { ... }
+     interface ge-0/0/2.0;
[edit protocols]
+   ldp {
+       interface ge-0/0/2.0;
+   }
+   lldp {
+       interface ge-0/0/2;
+   }
```

How do I change all of this from ge-0/0/2 to ge-0/0/3? With one very simple command:

```
[edit]
root@Router1# replace pattern ge-0/0/2 with ge-0/0/3

[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/3 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
```

```
[edit protocols]
+   mpls {
+       interface ge-0/0/3.0;
+   }
[edit protocols ospf area 0.0.0.0]
      interface ge-0/0/1.0 { ... }
+     interface ge-0/0/3.0;
[edit protocols]
+   ldp {
+       interface ge-0/0/3.0;
+   }
+   lldp {
+       interface ge-0/0/3;
+   }
```

Now are you starting to fall in love with Junos? I bet you are. I bet you are at least a *little* bit.

I have one final related command to show you, and it's a huge time-saver.

Imagine again the ge-0/0/2 isn't working. You want to test whether it's a problem with the physical port by moving the configuration over to ge-0/0/3, but this time, rather than actually move the configuration, you'd rather keep the original configuration on ge-0/0/2 as it is.

Junos offers a brilliant way to quickly configure this new interface: the `copy` command. Use this to copy entire stanzas from one place to another. You can then edit as necessary, for example disabling one interface while the other remains live to get around the fact that you now have the same IP on two interfaces:

```
[edit]
root@Router1# copy interfaces ge-0/0/2 to ge-0/0/3

[edit]
root@Router1#[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
+   ge-0/0/3 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
```

Modifying Configurations: Insert

A frequent problem that I find while teaching networking is that understanding one concept frequently involves the student knowing other topics first – but to know those topics, you first need to know about the original topic! This cyclic nature of knowledge depending on other knowledge leaves me saying to people: "remember this for now" and trust that we'll come back to it later.

The `insert` command is a great example of this, because to truly understand when and why it's used, you first need to know about firewall filters and routing policies. You'll learn about these in great detail later in the book.

Even though you've not been taught firewall filters yet, I very craftily placed a very simple firewall filter in the full configuration example configuration earlier in this chapter. Let's give it another look. Even if you don't fully understand firewall filters yet, seeing them now is a great way to lay some groundwork for when you come to grips with them later on. Some vendors call firewall filters *access lists*, and as you can see, they're slightly longer in Junos – but they're also written in plain English, making them very easy to understand:

```
root@Router1> show configuration | find firewall
firewall { /* OMITTED */ };

root@Router1>
```

Oh yeah, earlier on in this chapter I chose to omit the firewall hierarchy! Let's undo that:

```
[edit]
root@Router1# delete firewall apply-flags omit
```

Behind the scenes I've saved this change. Let's try again:

```
root@Router1> show configuration | find firewall
firewall {
    family inet {
        filter EXAMPLE_FIREWALL_FILTER {
            term DENY_SSH {
                from {
                    protocol tcp;
                    port ssh;
                }
                then {
                    discard;
                }
            }
            term ACCEPT_ALL_ELSE {
                then accept;
            }
        }
    }
}
```

If you read it slowly, you'll see I have one firewall rule called EXAMPLE_FIREWALL_FILTER. Specifically, this is a `family inet` filter, which means that it will only ever filter IPv4 traffic. This firewall filter has two terms, which are like two separate rules within the bigger rule itself. One of these terms denies SSH traffic. The other term accepts everything else. If this policy still doesn't make much sense even after reading it, don't worry. Chapter 6 explains how policies work in much greater detail.

For now, let's imagine we add an extra term to this filter, denying Telnet traffic:

```
[edit]
root@Router1# set firewall family inet filter EXAMPLE_FIREWALL_FILTER term DENY_
TELNET from protocol tcp

[edit]
root@Router1# set firewall family inet filter EXAMPLE_FIREWALL_FILTER term DENY_
TELNET from port telnet

[edit]
root@Router1# set firewall family inet filter EXAMPLE_FIREWALL_FILTER term DENY_TELNET then discard

[edit]
root@Router1# show | compare
[edit firewall family inet filter EXAMPLE_FIREWALL_FILTER]
       term ACCEPT_ALL_ELSE { ... }
+      term DENY_TELNET {
+          from {
+              protocol tcp;
+              port telnet;
+          }
+          then {
+              discard;
+          }
+      }
```

There's a problem with what we've done here.

You see, if you add a new term to an existing firewall filter, it's always added to the bottom of the filter. You'll soon come to learn that the terms in firewall filters are processed from top to bottom – and as soon as Junos matches the incoming traffic to a term, it stops processing the firewall filter. No further terms are checked.

In this scenario, the term ACCEPT_ALL_ELSE was previously at the end of the firewall filter. Our new term now goes at the bottom, below ACCEPT_ALL_ELSE. This means that ACCEPT_ALL_ELSE comes before the term DENY_TELNET. If you look carefully, you can actually see this in the output of our `show | compare` command above. Remember, policies are checked from top to bottom, and ACCEPT_ALL_ELSE is above DENY_TELNET. And because every single IPv4 packet will match the ACCEPT_ALL_ELSE term (apart from SSH, because that term came first), the DENY_TELNET term is never even checked.

To fix this, we need to `insert` our new term where it needs to be. Use this command to put your term above or below any other term:

```
[edit]
root@Router1# edit firewall family inet filter EXAMPLE_FIREWALL_FILTER

[edit firewall family inet filter EXAMPLE_FIREWALL_FILTER]
root@Router1# insert term DENY_TELNET before term ACCEPT_ALL_ELSE

[edit firewall family inet filter EXAMPLE_FIREWALL_FILTER]
root@Router1# top show | compare
[edit firewall family inet filter EXAMPLE_FIREWALL_FILTER]
       term DENY_SSH { ... }
+      term DENY_TELNET {
+          from {
+              protocol tcp;
+              port telnet;
+          }
+          then {
+              discard;
+          }
+      }
       term ACCEPT_ALL_ELSE { ... }
```

Can you see how this new DENY_TELNET term now sits in between DENY_SSH and ACCEPT_ALL_ELSE? Our firewall filter is now perfect. As well as `before`, you can also choose to insert a term `after` another term.

Were you ever told by other vendors that editing an access-list involved completely deleting it from the box, editing it in a text editor, and re-applying it, with your box essentially open to attack during the time that it's without the access-list? Well, I've got good news for you: with Junos, those days are over. Adding new pieces to the Junos equivalent of an access-list is very, very easy.

Modifying Configurations: Disable and Deactivate

We've seen almost every single command that's available to us in configuration mode: `set`, `edit`, `copy`, `delete`, `run`, `insert`, `top`, `up`, `wildcard`, `help`... And we've covered a huge amount in this chapter. Amazingly, there's still more. With Junos, you have true power at your fingertips. Here's the full list of commands you can type in configuration mode:

```
[edit]
root@Router1# ?
Possible completions:
  <[Enter]>            Execute this command
  activate             Remove the inactive tag from a statement
  annotate             Annotate the statement with a comment
  commit               Commit current set of changes
  copy                 Copy a statement
```

```
  deactivate           Add the inactive tag to a statement
  delete               Delete a data element
  edit                 Edit a sub-element
  exit                 Exit from this level
  extension            Extension operations
  help                 Provide help information
  insert               Insert a new ordered data element
  load                 Load configuration from ASCII file
  prompt               Prompt for an input
  protect              Protect the statement
  quit                 Quit from this level
  rename               Rename a statement
  replace              Replace character string in configuration
  rollback             Roll back to previous committed configuration
  run                  Run an operational-mode command
  save                 Save configuration to ASCII file
  set                  Set a parameter
  show                 Show a parameter
  status               Show users currently editing configuration
  top                  Exit to top level of configuration
  unprotect            Unprotect the statement
  up                   Exit one level of configuration
  wildcard             Wildcard operations
```

You'll recognize `disable` from earlier in this chapter, when we showed you how to admin-down an interface. A handy troubleshooting tip is to try shutting down an interface, and then bringing it back up. To do that, simply configure the interface to be disabled, save your changes, then roll back to the last configuration and save it again. Sorry, what's that? What do I mean by rollback to the last configuration? Hold that thought for a moment.

An alternative to `disable` is `deactivate`. The difference is that `deactivate` actually makes the router pretend that your configuration isn't even there to begin with. To show the effect of this command let's quickly look at our interfaces as they stand right now:

```
root@Router1> show interfaces terse | match ge
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     10.10.13.1/24
ge-0/0/2                up    up
ge-0/0/2.0              up    up   inet     10.10.14.1/24
{snip}
```

Notice that ge-0/0/1 has both a physical and a logical entry. Now let's try deactivating interface ge-0/0/1.

```
[edit]
root@Router1# deactivate interfaces ge-0/0/1

[edit]
root@Router1# show | compare
[edit interfaces]
!    inactive: ge-0/0/1 { ... }
```

Behind the scenes, I'm going to save this command. Let's check our interfaces again:

```
root@Router1> show interfaces terse | match ge
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     10.10.12.1/24
ge-0/0/1                up    up
ge-0/0/2                up    up
ge-0/0/2.0              up    up   inet     10.10.14.1/24
{snip}
```

The physical interface is still there but the logical interface has completely gone! The `disable` command kept it there and just showed it as admin-down. By contrast, `deactivate` makes it look like there was never any configuration there in the first place.

This command is particularly useful if you want to add any configuration in advance, but leave it in a deactivated state. You can then simply issue an `activate` command when you're ready for your configuration to go live.

Even with a four-million IQ, as certified by both NASA and the FBI, I still regularly forget the difference between `disable` and `deactivate`. Try to remember that `disable` means admin-down. By contrast, use `deactivate` when you want to pretend for a while that a piece of configuration never even existed.

Commenting and Protecting Your Configuration

If you've been in networking for a while, you'll know the pain of logging on to a new router and seeing a long configuration. Take time to read the config, and you'll probably understand what the router is doing but not **why** it's doing it. The best you can hope for is that someone has left a description on an interface to tell you what it connects to.

Junos certainly offers this, and you configure it like this:

```
[edit]
root@Router1# set interfaces ge-0/0/1 description Connection to ISP via Evil Internet Inc
```

Junos also allows you to leave comments throughout the configuration, in the form of the `annotate` command. Use this to leave a comment at the start of every top-level stanza. For example, imagine that I wanted to make sure people knew that only I was allowed to make protocol changes on this box. I could annotate the `protocols` stanza like so:

```
[edit]
root@Router1# annotate protocols Only Chris Parker can configure this section!
```

As a result, when someone views the full configuration, they'll see the comment. I've removed some lines in the output below, to get right to the point:

```
root@Router1> show configuration
{snip}
routing-options {
    autonomous-system 64512;
}
/* Only Chris Parker can configure this section! */
protocols {
    ospf {
        area 0.0.0.0 {
            interface ge-0/0/0.0;
            interface ge-0/0/1.0;
        }
    }
}
firewall {
    family inet {
{snip}
```

In labs, students often leave themselves comments at the `system` stanza because the comment appears at the very top of their configuration. This is extremely useful for leaving yourself notes about what you've done so far, so you can more easily pick up where you left off on another day. In the real world, I occasionally see engineers leaving their name and date above a new stanza they've added.

In my example above, I told people not to edit the protocols hierarchy. But of course, that relies on someone actually seeing my comment – and even then, it relies on them listening to me!

I chose this example on purpose because you're probably thinking that leaving a note isn't the best way to ensure that someone doesn't accidentally delete all your BGP configuration. Wouldn't it be handy if it was possible to lock down certain pieces of configuration so that they couldn't accidentally be deleted? Yes, it would. And that's why the wizards behind Junos did exactly that, and added this command:

```
[edit]
root@Router1# protect protocols
```

Once that's saved, if someone tries to delete this interface, their commit will fail:

```
[edit]
root@Router1# delete protocols
warning: [protocols] is protected, 'protocols' cannot be deleted
```

This is another command that should definitely be added to your arsenal of best practices. Consider protecting all your protocols, and all your core-facing interfaces. Use `unprotect` to undo this command. The business case for doing this is undeniable.

Managing and Saving Configuration Files: Commit, Commit and Quit, Commit Confirmed

Okay, you're now an expert at making changes, and at deleting those changes. You can edit, move, rename, replace, copy, annotate, and protect. The only thing left to do is to actually save our work. Are you ready to bring all your ninja training to the table?

I've put the IP and IPv6 address configuration back on to ge-0/0/2. This interface is blank in the active configuration, and now has these proposed changes in our candidate configuration:

```
[edit]
root@Router1# show | compare
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
+   }
```

Let's talk about how we save configurations in Junos. Be warned: what you're about to read might revolutionize the way you work.

At a basic level, as we saw earlier, you can save by simply typing `commit`. This saves your work, and keeps you in configuration mode:

```
[edit]
root@Router1# commit
commit complete

[edit]
root@Router1#
```

Alternatively, are you finished making configuration changes? In that case, type `commit and-quit` instead. This saves your config, then takes you back to operational mode:

```
[edit]
root@Router1# commit and-quit
commit complete
Exiting configuration mode

root@Router1>
```

The ability to save all changes at once is already so much more reassuring than a CLI where each individual command goes live as-and-when we type it. Now, let's take it to the next level.

With other vendors, you might have been given advice along these lines: if you're configuring a device remotely, and you accidentally configure something that makes you lose access to it, you can regain control by making sure that you type a command which automatically reboots after a few minutes, before you start making your changes. That way, if the worst case scenario comes to light and you do indeed lose access, no worries: the box will reboot, and you'll go back to the working configuration that was on it before you started making changes.

Obviously no one is denying that rebooting a device, and causing a potential 10-minute outage while you wait for it to reboot and re-calibrate, is an absolutely brilliant way to undo your change. Truly fantastic. An undeniably great process. Everyone loves downtime! Brilliant and perfect advice, and not in any way terrible or awful, or bad. You won't find an engineer in the world who thinks this is anything other than the best possible way to deal with this situation. But: what if there was A Better Way™?

If you're making a configuration change and you suspect in any way that your change might cause an unexpected problem, save your work with the command `commit confirmed`. This tells Junos to automatically roll back to the previous configuration after 10 minutes. Now, if you do accidentally make a fatal mistake that takes away your management access, then at least the box is still operational, and can carry on processing traffic.

Perhaps ten minutes is too long to wait? Or perhaps the change might actually take down the ability for the box to process transit traffic? In that case, type `commit confirmed 2`, or however many minutes you like, from 1 all the way up to 65,535 minutes, to get the quickest rollback suitable for your change. Though why you'd want to wait 65,535 minutes for your change to roll back is anyone's guess. Some people like to live dangerously, I suppose. Anyway, here's what it looks like:

```
[edit]
root@Router1# commit confirmed 3
commit confirmed will be automatically rolled back in 3 minutes unless confirmed
commit complete

# commit confirmed will be rolled back in 3 minutes
[edit]
root@Router1#
```

Did your change go well? You're still in configuration mode – so just type commit to truly save your changes. Alternatively, just wait 3 minutes, and your command prompt will show you this output. Press Return when you see it to get back to the CLI:

```
[edit]

Broadcast Message from root@Router1
        (no tty) at 0:39 UTC...

Commit was not confirmed; automatic rollback complete.
```

What if your change didn't go well? How do you get back to your previous config? Do you just have to wait 10 minutes, or however long you've specified it for? Actually, no: you simply have to type `rollback 1` to re-load the previous configuration! If by chance it isn't clear why that command fixes your problem, hold that thought – because in a moment we're going to look at our commit history, at which point it will be a lot clearer why this works.

Managing and Saving Configuration Files: Commit Check, and Commit Synchronize

What if we want to verify that our configuration is good before we save it?

Well, as it happens, Junos checks our configuration line-by-line, word-by-word, as we type it. Every single new word we type is verified for correct and proper syntax. This is great! It significantly reduces the chance of errors.

However, these checks are purely for syntax. What if we do something like add a named firewall rule to an interface when we haven't yet configured the firewall rule itself? Well, Junos helps us out once more: it won't allow us to commit the change. Again, this massively reduces the number of user-generated configuration errors you'll find in production environments.

Let's see how we might add a firewall filter to an interface when we haven't configured the rule itself:

```
[edit]
root@Router1# set interfaces ge-0/0/1 unit 0 family inet filter input FAKE_FIREWALL_FILTER

[edit]
root@Router1# commit
[edit interfaces ge-0/0/1 unit 0 family inet]
  'filter'
    Referenced filter 'FAKE_FIREWALL_FILTER' is not defined
error: configuration check-out failed
```

Now that you see how helpful Junos can be, consider this scenario. Imagine that you wrote out, in advance, thirty lines of new configuration. You're not going to deploy this new configuration for a couple of weeks. Still though, you'd love to be able to check in advance that your configuration actually makes sense and won't cause any errors when it is deployed. This ability to verify your work without actually saving it sounds very handy – it would certainly prevent problems at 2 a.m. when the engineer tasked with entering your change suddenly finds that your proposed change isn't quite correct!

In Junos, these checks are entirely possible.

First of all, you can safely add all your configuration without it going live, due to the concept of the candidate configuration. Then, instead of typing `commit`, you can

type `commit check`. As you'd imagine, this checks your candidate configuration for errors, without actually committing the config. Typing this command will give you exactly the same error as we saw in the previous example, which means you can go away and fix your config in advance. What a lifesaver! Just remember to type `rollback` to scrap all those changes before you leave configuration mode.

Just to be clear: if you've configured an incorrect IP address on an interface, `commit check` won't spot that. There's no way for the device to know if your IP address is correct or not. But `commit check` is perfect for checking that you haven't accidentally told BGP to use a routing policy that doesn't exist, or that you haven't accidentally configured an interface as both a Layer 2 switchport and a Layer 3 routed port. I use that last example on purpose, because on some more advanced Juniper devices it's actually possible to do exactly that. It just needs to be done in the right way, and `commit check` will readily tell you if you've done it correctly.

By the way, do you remember in Chapter 1 when Martin taught you about devices with multiple routing engines for redundancy? Changes on these devices need to be committed in a special way, because by default, typing `commit` only saves your configuration to the primary RE. That's no good. You need your backups to be exactly the same between both REs. It's useless having redundancy if your backup isn't the same as your primary!

The less-smart way of saving your change on both REs is to use the command `commit synchronize` every time you save your work. This is fine – as long as you and your colleagues always remember to do it.

The four-million-IQ way of saving your change on both REs is to add the command `set system commit synchronize` as one of the very first things you do when you take your new multi-RE Junos device out of the box. You'll only then need to do a `commit synchronize` the one time, to get this command onto both REs. After that, you and your colleagues will just need to type `commit` like you would on any other box. Your candidate configuration will then successfully be saved to both automatically.

Comparing, and Reverting to, Previous Configurations

By default, Junos keeps fifty previous versions of your configuration on modern Juniper devices! When more than 50 versions exist, Junos simply discards the oldest one. In later chapters of this book you'll learn how you can also automatically save every configuration to an external FTP or SCP server, so you can keep a full history of every change ever made.

To see all the currently stored historical configurations, type `show system commit`. (There is an outside chance that some of the usernames in the logs below may have been edited by me. I'll leave it to you to decide):

```
chris.parker@Production_Box> show system commit
0   2020-04-17 13:24:38 BST by ryan.gosling via cli
1   2020-04-17 13:24:16 BST by ryan.gosling via cli commit confirmed, rollback in 10mins
2   2020-04-16 15:21:53 BST by brad.pitt via cli
3   2020-04-16 15:21:38 BST by brad.pitt via cli commit confirmed, rollback in 10mins
4   2020-04-14 11:53:30 BST by chris.parker via cli
5   2020-04-14 11:47:45 BST by chris.parker via cli commit confirmed, rollback in 10mins
6   2020-04-08 16:32:52 BST by beyonce via cli
7   2020-04-08 16:32:37 BST by beyonce via cli commit confirmed, rollback in 10mins
8   2020-04-08 16:12:10 BST by ryan.gosling via cli
9   2020-04-08 16:11:04 BST by ryan.gosling via cli commit confirmed, rollback in 10mins
10  2020-04-08 10:44:35 BST by beyonce via cli
11  2020-04-08 10:44:20 BST by beyonce via cli commit confirmed, rollback in 10mins
12  2020-04-07 12:07:09 BST by chris.parker via cli
13  2020-04-07 12:06:40 BST by chris.parker via cli commit confirmed, rollback in 2mins
14  2020-04-07 11:38:30 BST by brad.pitt via cli
15  2020-04-07 11:33:21 BST by brad.pitt via cli commit confirmed, rollback in 10mins
```

You can see the date that the configuration was deployed (one of many reasons to make sure your system time is accurate), and the name of the engineer who made the change. If any comments were added at the time of commit, you'll see them here, too.

Notice that configuration 0 is the most recent configuration – in other words, your active configuration. Configuration number 1 is the previous config; configuration number 2 is the one before that; and so on. Typing `rollback` is really the equivalent of typing `rollback 0`. As we'll see in a moment, you can roll back to any of these configurations simply by saying the number you want to roll back to.

Notice as well that whenever you do a `commit confirm`, and then follow it up with a final `commit`, each of these individual commands creates a new backup configuration. Imagine that you'd performed a `commit confirm` and decided that the changes weren't to your liking. Sometimes people think that they have to wait 10 minutes for their changes to roll back. But actually, you can just reload the previous configuration, and save it again. This is essentially the same as undoing your change – you are literally going to roll back to the previous configuration! Type `rollback 1` to bring the previous config back into the candidate config, at which point you can `commit` once again.

Earlier we learned how to `annotate` stanzas inside the configuration. This time though, let's leave a comment on the actual configuration file to say what changes were made when my colleagues were looking at this list of configurations. Sounds like another good thing to make a best practice in your organization! This time I'm going to do it, and view the results, on my lab device. Type `commit comment XYZ` to leave a comment that says XYZ. Replace XYZ with anything you like:

```
[edit]
root@Router1# commit comment Chris Parker, adding BGP to the box
commit complete
```

Let's look at our commit history again and see the results of this comment. Notice my comment underneath revision 0 of the config:

```
root@Router1> show system commit
0   2019-12-19 00:52:39 UTC by root via cli
    Chris Parker, adding BGP to the box
1   2019-12-19 00:39:06 UTC by root via other
2   2019-12-19 00:35:12 UTC by root via cli commit confirmed, rollback in 3mins
3   2019-12-19 00:32:12 UTC by root via cli
{output edited for brevity}
```

This is really handy for quickly identifying what changes were made in a particular commit. There really isn't a reason to *not* do this every time you commit a change on a production box.

As you can imagine, it's possible to roll back to any of these configurations. If you've added comments, you'll find it extremely easy to know precisely which configuration you need to roll back to.

Let's imagine that I wanted to roll back to configuration number 5 on my lab router. Is it difficult? No way! In configuration mode, simply type:

```
[edit]
root@Router1# rollback 5
load complete
```

That's it! That's literally it. It's as easy as that.

It's important to understand that this `rollback 5` command doesn't save your work – it just loads configuration number 5 into the candidate configuration. This means you can do your friendly neighborhood `show | compare` command to see what changes are going to be made, and in addition you can add or remove anything extra before you actually save your work.

It's great that you can do a `show | compare` to see the differences between your current active and candidate configuration – but what about if you want to compare your current active configuration to an older configuration, without first importing it into the candidate configuration? Again, this is easy. Simply type `show configuration | compare rollback 6` to compare your current active configuration with – you guessed it – the sixth previous version of the configuration!

```
[edit]
root@Router1# show | compare rollback 6
[edit interfaces]
+   ge-0/0/2 {
+       unit 0 {
+           family inet {
+               address 10.10.14.1/24;
+           }
+           family inet6 {
+               address 2001:db8:0:14::1/64;
+           }
+       }
```

Managing and Saving Configuration Files Using the Save Command

You now should be comfortable with the idea of saving configurations via the `commit` command. You're also savvy about viewing the past 50 configurations, and comparing them with the active configuration. Junos has offered this functionality for years; and even now it is still far more than some other vendors offer.

But as always, Junos doesn't stop there: you can also save a backup of your candidate configuration – that's the candidate, not the active – at any time, to a named file of your choice. Type this command to save your file to your home directory:

```
[edit]
root@Router1# save BACKUP_CONFIG_FILE.txt
Wrote 78 lines of configuration to 'BACKUP_CONFIG_FILE.txt
```

The .txt extension isn't mandatory, but I like to add it in case the file is ever exported off of the box. You can also specify a particular directory to save it to.

In your home lab, this is a fantastic way to store a base configuration that you can always return to. It allows you to add protocols, delete interfaces, mess up and break everything up as much as you like in the safety of your laboratory – and then, just console in and roll back to this file. Bear in mind: a consequence of the fact that we're saving the candidate configuration here is that if you have pending changes waiting to be saved, and you run this command, it will also save your uncommitted changes in the file you create.

This `save` command can be used in operational mode, too; you can send the output of any command to a text file. Just add a pipe (`|`), and tell Junos to save it. We can use this to create a backup of our active configuration at any time we like:

```
root@Router1> show configuration | save SAVED_CONFIG.txt
Wrote 78 lines of output to 'SAVED_CONFIG.txt'

root@Router1> file list

/root/:
.cshrc
.login
.profile
SAVED_CONFIG.txt
```

Once you've saved a file, you can then view it, copy it, delete it, move it, compress it, compare it to other files, and pretty much anything else you hoped to be able to do. Type `file show FILENAME` to view the contents of any text files on your device.

So far we've learned approximately fifty trillion different ways that we can save configurations. And this book still isn't done. In Chapter 3 you're going to learn all about something called configuration archiving – in other words, how to set up Junos to automatically push configurations to an external FTP or SCP server. But it's no good saving configurations if you can't load them, right?

Loading Configurations

All of the configuration commands we've learned so far involve typing `set` commands. But here's a puzzle: these configuration files we've been saving don't use `set` commands. Instead, these config files are stored in hierarchical format. We said earlier that we don't type commands in hierarchy format, which begs the question: even if we do have saved backup versions of these configurations, how on earth do we get them back on the box? Don't worry: the answer is surprisingly simple.

Junos does indeed give you the option to enter commands in hierarchy format, which means you can either just import a configuration file directly into your candidate configuration, or you can copy-and-paste the hierarchy configuration into the terminal. In this section we're going to learn both methods.

To import your configuration from a file, use the command `load override YOUR_FILENAME_HERE.txt`. The word `override` in that command is important. With this keyword, we're telling Junos to completely scrap the current candidate configuration and replace it with this file. It's the equivalent of just typing `delete` in configuration mode, to delete absolutely everything at every hierarchy level, then re-configuring the device from scratch.

Let's look at an example, to make it clearer. I'm going to make some tiny changes to highlight the behavior of this command.

On my vMX I currently have configured three interfaces: ge-0/0/0, ge-0/0/1, and ge-0/0/2. Let's delete ge-0/0/0 from our candidate configuration. Let's also add a new interface, ge-0/0/8. I won't put any IPs on this fake interface – I'll just add a new description. You might do something like that in the real world to reserve a port on a large device, in advance of some installation work.

Once we've done that, let's save the candidate configuration to a file called DELETED_GE0.txt. I'm not committing these changes – I'm just writing this new candidate configuration to a file:

```
[edit]
root@Router1# delete interfaces ge-0/0/0

[edit]
root@Router1# set interfaces ge-0/0/8 unit 0 description FAKE INTERFACE

[edit]
root@Router1# save DELETED_GE0.txt
Wrote 76 lines of configuration to 'DELETED_GE0.txt'
```

You'll find no mention of ge-0/0/0 inside this file – which gives us the perfect opportunity to see what happens when we replace an existing configuration with this new one.

Let's roll back, so all those changes I made are gone. After the rollback, our ge-0/0/0 interface is once again in both the active and candidate configurations:

```
[edit]
root@Router1# rollback
load complete

[edit]
root@Router1# show | compare

[edit]
root@Router1#

[edit]
root@Router1# show interfaces ge-0/0/0 | display set
set interfaces ge-0/0/0 unit 0 family inet address 10.10.12.1/24
```

Remember that when you see the `show | compare` command come back with nothing, it means your active and candidate configurations are exactly the same. Remember, as well, that the behavior of the `show interfaces` command in configuration mode is different from what we'd see using this same command in operational mode. It doesn't show us the interface status – it shows us the interface configuration.

The file DELETED_GE0.txt is a full configuration in hierarchy format, and it's almost exactly the same as my current active configuration – apart, of course, from those interface changes we made. Let's use this file to see the behavior of the `load override` command. I'm going to ask Junos to completely scrap the current candidate config, and replace it with my new file:

```
[edit]
root@Router1# load override DELETED_GE0.txt
load complete

[edit]
root@Router1# show | compare
[edit interfaces]
-   ge-0/0/0 {
-       unit 0 {
-           family inet {
-               address 10.10.12.1/24;
-           }
-       }
-   }
+   ge-0/0/8 {
+       unit 0 {
+           description FAKE INTERFACE;
+       }
+   }
```

Nice! It worked perfectly. Interface ge-0/0/0 was missing from my text file, and as such, importing that file means that we're removing ge-0/0/0 from the router.

So you can use `load override FILENAME` when you want to scrap any and all configuration currently on your device, and replace it entirely with the contents of the file of your choice. You use this command when the file is stored on the router itself. What about if the configuration is instead stored on your local computer? For example, what if you have a new Junos device on your desk to replace one that's

died, and you have a backup configuration in a text file stored on your local PC? Easy answer: you can paste it into your console or SSH/Telnet session using the command `load override terminal`. This changes your command prompt to look like this:

```
[edit]
root@Router1# load override terminal
[Type ^D at a new line to end input]
```

…at which point you can just paste in your backup configuration and press Ctrl-D when you're done. D for done! You'll be taken back to the usual configuration mode prompt where you can save your new config, or indeed make extra changes using `set` commands. I've used the `load override terminal` command many times over the years, and I'm sure you will too. (And as always, all of these commands are hierarchy aware. You can save just a portion of the hierarchy to a file, load in just a portion of the hierarchy, or whatever you'd like!)

There are a few other ways to `load` a file besides `override`, and there are two in particular that you'll want to know about:

- `merge`: This method keeps the candidate configuration as it is but still imports your new configuration. In other words, using `load merge` merges the two configs together. Some configs will be overwritten with your new changes; other pieces will stay the same.
- `replace`: This method isn't quite as intuitive as you might think. The `override` command fully replaces the existing config with your new one. And moreover, Junos allows you to edit your saved configuration to add `replace` tags throughout it by using the `load replace` command, which tells Junos to parse your saved configuration, and to only replace the sections that you've tagged with `replace`. This is good for when you have a large configuration, but you only want to import certain sections of it.

In the real world, engineers on smaller Junos devices tend not to store a variety of configurations locally on the device, preferring to `load override terminal` whenever necessary. However, on larger boxes it's fairly common to move files to and from the box, and to load them directly. For that reason, make an effort to save and load some active and candidate configurations in your lab. Make yourself a base configuration that you can always roll back to. Move files around, delete them, and have a play with the options. If you're truly confident in the commands, you'll feel even more confident as an engineer operating on a production network.

Before we move on to the graphical interface here's one little pro-tip for you, to make your CLI even easier to use. Most vendors use a default number of characters that your command line can show at once, and almost every vendor tends to default to 80 characters. If you type more than that number of characters, you'll often find that the CLI shifts the text along, which can lead to something like this:

```
[edit]
```

```
root@NewRouter# ... unit 0 family inet address 192.168.10.1/24
```

As a force of habit, whenever I log on with the CLI, the very first thing I do in operational mode is type `set cli screen-width 500`. This gives you a much longer CLI, and more readable results:

```
root> set cli screen-width 500
Screen width set to 500

root@NewRouter> configure
Entering configuration mode
The configuration has been changed but not committed


[edit]
root@NewRouter# set interfaces ge-0/0/4 unit 0 family inet address 192.168.10.1/24
```

Remember to do this particular set commands in operational mode, not configuration mode!

J-Web: The Junos Graphical Interface

With all this power at your fingertips, you can no doubt see why many engineers favor the CLI over a GUI.

Nevertheless, it's useful to have a familiarity with the GUIs of your vendors of choice. For example, one day in the future you could well be deployed to set up a small office network at an organization that employs some excellent IT staff who know the philosophy of networking, but have never dug deep into any CLIs. In this situation, giving them access to something like J-Web is perfect. By showing them the options available to them, you can empower trusted users to manage their own network, greatly reducing the number of simple change tickets that will come through to you.

Now, an advantage of this book being an Ambassador guide is that I can let you in on an Ambassador secret: historically, people haven't enjoyed using J-Web. Previous versions of it were, unfortunately, not as responsive or smooth an experience as one might hope. I say this because I have good news for you: in recent years, J-Web has had a complete make-over. It now has the same lightweight, elegant aesthetic as Security Director (the module in Juniper's Junos Space product, a platform that lets you manage multiple devices in one place).

If you happen to have used J-Web in the past, try giving it another go on boxes running Junos code from 2019 onwards. The new interface is clean and responsive, and should give you and your users a very positive experience.

Accessing J-Web

J-Web functionality comes preconfigured out of the box on some smaller Junos devices. Remember we noted at the start of this chapter that some interfaces on certain Juniper products come pre-enabled with DHCP, allowing you to gain access to the graphical interface quickly and easily. The SRX550 firewall is one example of such a box. (visit the Juniper TechLibrary for this step-by-step breakdown of the procedure: https://www.juniper.net/documentation/en_US/release-independent/junos/topics/task/operational/services-gateway-srx550-from-cli-locally-connecting-usb.html). Once you've gained an IP and have navigated to https://192.168.1.1 in your web browser, you'll be taken through a setup wizard that allows you to configure some essential features such as the hostname and the root password, before going to configure the interfaces themselves.

On all other platforms, J-Web needs to be actively turned on, which means you'll need to console in via the CLI first and set an initial configuration – root password, an IP to log onto, and so on. You can then turn on J-Web using this configuration mode command:

```
[edit]
root@Router1# set system services web-management https interface ge-/0/0/0
```

You should now be able plug a cable between your computer and the port you gave an IP to, and browse to the J-Web interface via HTTPS. If you've done it right, you'll see a log in screen as in Figure 2.4.

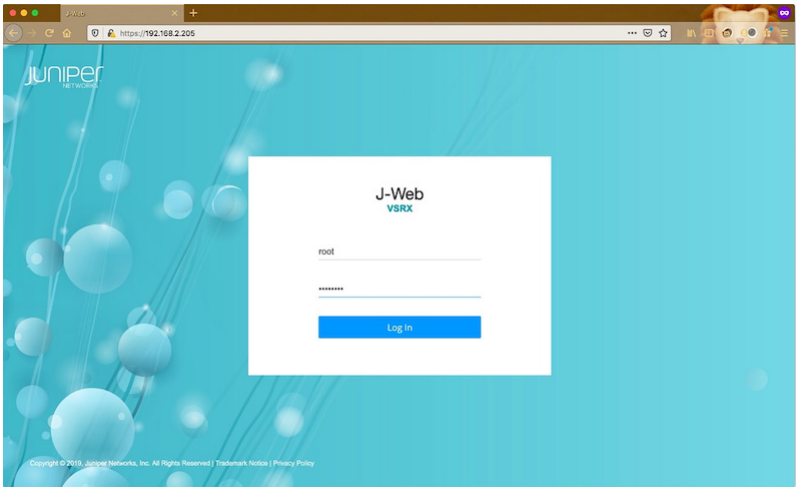


Figure 2.4 J-Web Log In

Navigating J-Web

Entering your username and password in Figure 2.4 will reward you with the following screen, shown in Figure 2.5, which allows you to configure basic settings. You'll see that it's easy for someone who knows no Junos to just jump right in and configure things like interfaces and the date/time.

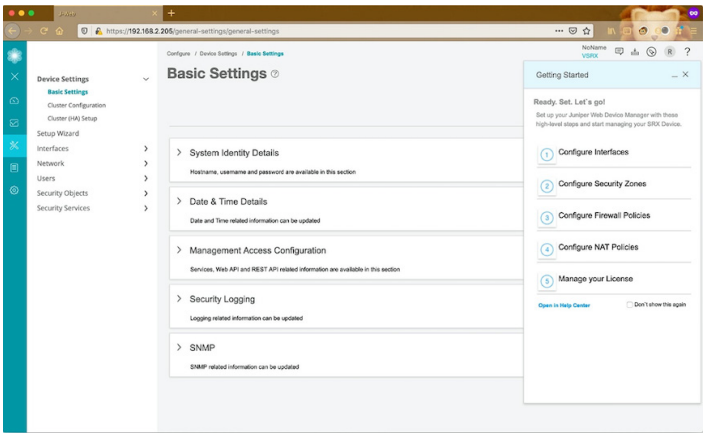


Figure 2.5 Configure Basic Settings

On the left-hand side of Figure 2.5 you'll notice a light blue navigation bar with various icons. You can hover over these to expand them, at which point you'll see that each icon takes you to a different section of J-Web. For example, clicking the speedometer icon takes you to the Dashboard for all the high-points of your box, like basic system information (hostname, serial number, etc.), and memory/CPU usage. If you click on the spanner-and-screwdriver icon in that blue bar on the left, as we've done in Figure 2.6, a menu will pop up allowing you to configure the box.

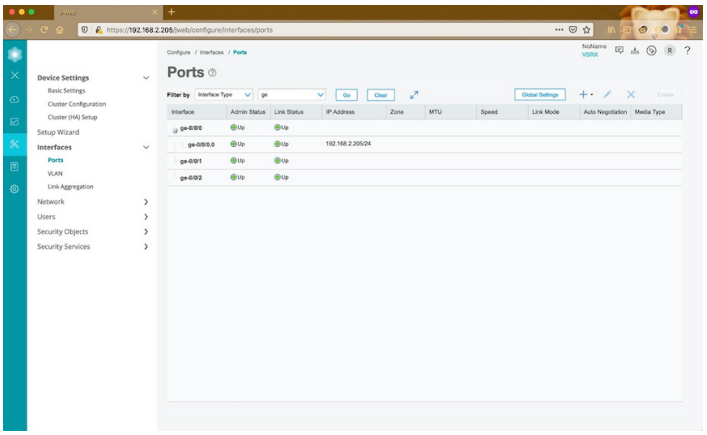


Figure 2.6 Drilling Down into J-Web

Figure 2.6 also shows the Interfaces > Ports section. You'll notice that there are many more options in the menu. Quite a lot of these options involve concepts that this book hasn't introduced yet, such as VLANs and routing. Others are concepts that you'll learn in detail as you go further into your career, such as chassis cluster and class of service. It's clear that you can do some very interesting stuff through the graphical interface!

Only a few interfaces are currently configured in Figure 2.6. Interface ge-0/0/0 has been sub-interfaced and given an IP address. You can add new interfaces by clicking the + icon near the top-right of the page, or you can edit any interface by clicking it and then clicking the pencil symbol in the top-right to edit it. If you do click the pencil symbol, you'll be treated to a pop-up like the one shown in Figure 2.7.

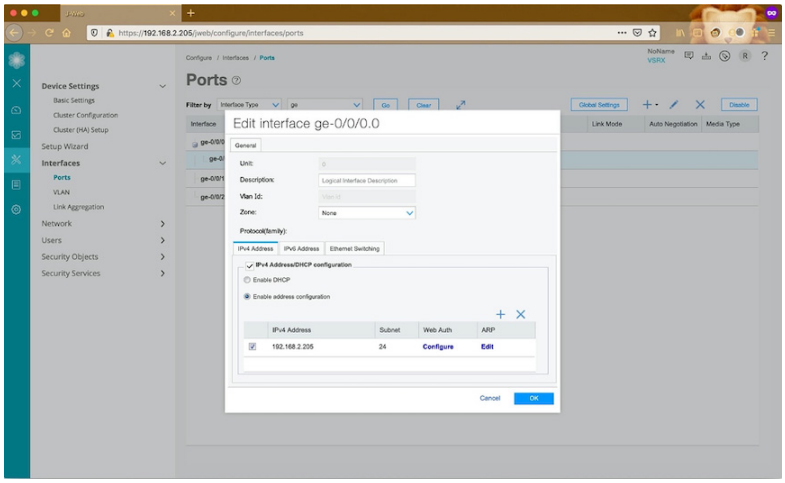


Figure 2.7 *Editing Interface Example*

Notice that you can add IP addresses, and put the interface into a security zone. In short, on SRX firewalls you can put interfaces into security zones and then make security policies that allow or deny traffic between zones. Changes are saved to the candidate configuration whenever you click OK. Then, just like on the CLI, you can click the icon of the arrow pointing downwards in the top-right corner of the screen to bring up a menu to commit your changes, commit confirm, compare your changes to the current config, and more, as in Figure 2.8.

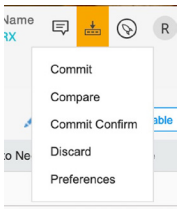


Figure 2.8 *Commit*

Let's now go back to that blue menu bar on the left, and click on the cog icon, to bring up the Administration menu (Figure 2.9).

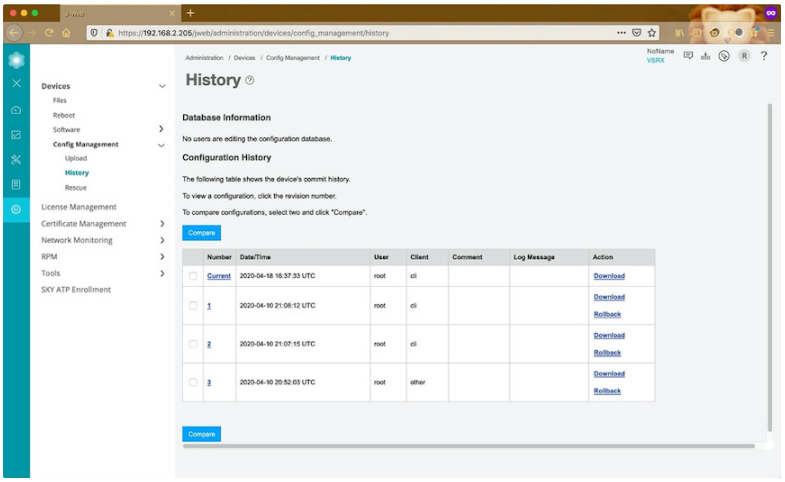


Figure 2.9 Administration Tasks

I've navigated to Device > Config Management > History, where you can see a full list of all the configurations. I can even compare any two configs to each other. Sure, it's not as fun as doing it on the CLI, but if you've got some engineers who don't know Junos, this GUI is a very handy tool to give them access. You can do a whole heap of handy stuff in the Administration section of J-Web, like upgrading Junos. Doing this via the CLI isn't difficult once you know what you're doing but I personally think it's even easier if you just click a button and navigate to a Junos image saved on your desktop. Go to Device > Software > Upload Package, and as shown in Figure 2.10, you'll see it's as easy as pie.

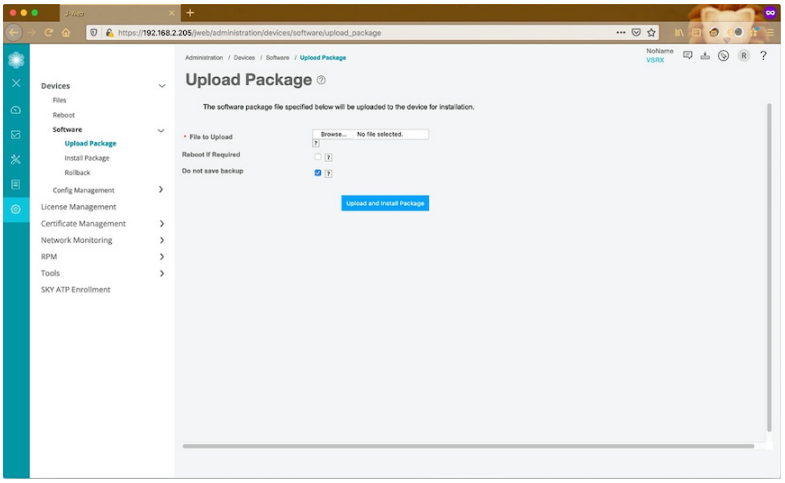


Figure 2.10 Easy as Pie

Finally, you can even troubleshoot via the GUI as shown next in Figure 2.11.

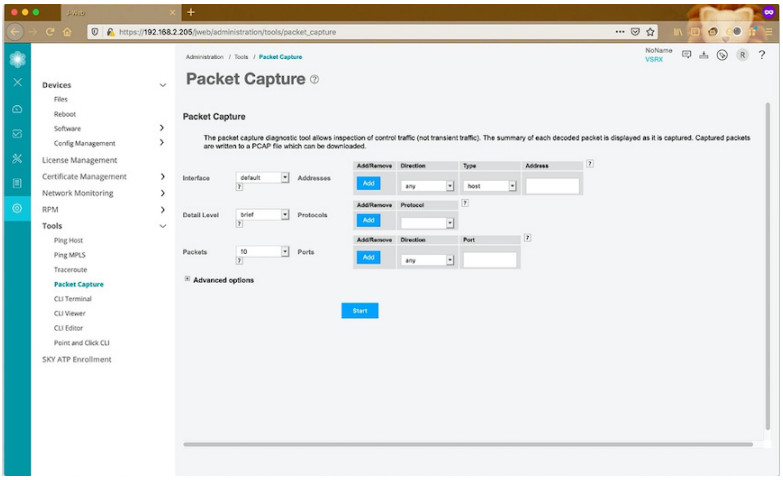


Figure 2.11 *Troubleshooting on J-Web*

You can see ping and traceroute in the menu on the left. But perhaps one of the most interesting options is the ability to perform packet captures! You can do this on the CLI too, but again, the ability to do this in the GUI and save the packet capture directly to your desktop is incredibly useful. Go to Tools > Packet Capture, choose your interface of interest, and optionally go into the Advanced Options to customize what you'll receive.

It's worth mentioning that this packet capture doesn't capture every single packet on the wire. Instead, it captures control plane traffic – in other words, it captures traffic destined to the box itself, or traffic being generated by the box. If you want to grab stuff like BGP and OSPF packets, this is the place to be. You'll learn more about this concept in later chapters, when my colleagues teach you how to use the `monitor` command.

We've only scratched the surface of what's possible with J-Web. There's a Monitoring section that lets you view things like interface statistics and VPN sessions; there's a Reports section that lets you view traffic logs; and of course wizards that guide you through navigating some of the more complicated elements of creating a working Junos device.

Summary

Wow, what a chapter!

You've learned how to navigate around the Junos CLI and even the J-Web graphical interface. You've learned how to use `show` commands in operational mode to see almost everything you'd want to know about a Junos device and the protocols it's running; you've seen how to go into configuration mode and make changes; you've been shown that there's more than just one way to save a configuration; you've watched as we rolled back to previous configurations, moved things, renamed things, merged things, and more.

With all that in mind, I hope you'll allow me to repeat myself for one final time, because it truly is the most important lesson of this chapter – the only way to really learn any of this is to get hands-on with Junos.

You could read this chapter a dozen times and still only take in 30% of it. But if you get hands-on in a lab and actually type these commands yourself, you'll come away with a supreme confidence because you'll genuinely have learned it. Well, that is unless you hit the Christmas Gin a bit too hard, as this author regularly does, in which case you'll come back from your holiday break to discover that you've forgotten everything you once knew about computers.

But forget I said that. You might be wondering how you can actually practice all of this if you're a person of limited financial means. No worries: at the back of this book we've included an Appendix that shows you three different ways to create your own virtual lab entirely for free. If you've got the time, go straight to the Appendix and try to set up a virtual lab. That way, as you read the following chapters written by my esteemed Juniper Ambassador colleagues, you can type the commands yourself as you go.

Re-read this chapter if you need to. There's a lot to take in but if you commit to practicing in a lab, you'll have every single piece of it mastered in no time. Have fun labbing it up, do your best, and thank you for enduring my enthusiasm!

Chapter 3

Junos Configuration Basics

By Christian Scholz, Paul Clarke, and Peter Klimai

Juniper devices come from the factory with Junos installed on them. When you power on a Juniper device, all software will start automatically. However, you will need to configure the software so that the device is ready to participate in your specific network design.

To configure Junos, you must specify a hierarchy of configuration statements that define the preferred software properties. You can configure all properties of Junos, including interfaces, general routing information, routing protocols, and user access, as well as some system hardware properties. After you have created a candidate configuration, you commit the configuration to be evaluated and activated by Junos.

This chapter covers various Junos configuration basics to help you get started with your network deployment.

Factory Default State

The default factory configuration which contains the basic configuration settings is the configuration on the device when the device is first powered on. This is the configuration presented to you when you first log in.

If for any reason the current active configuration fails or needs to be overwritten, you can revert to the default factory configuration.

The `load factory default` command is a standard Junos configuration command. This configuration overrides existing configuration with factory default configuration and replaces the current active configuration.

To revert to the factory default configuration, type this command in configuration mode:

```
user@Junos# load factory default
```

NOTE Because the `load factory default` command requires a commit you will need to specify a root-authentication password, otherwise the commit will fail, as shown here:

```
user@Junos# commit check
[edit]
  'system'
    Missing mandatory statement: 'root-authentication'
error: configuration check-out failed: (missing mandatory statements)
```

WARNING Make sure you have console access either locally or remotely via an out-of-band network because after you commit the factory default configuration you will be isolated from the outside world.

The `load factory default` command should not be confused with `request system-zeroize` Junos command.

The `request system-zeroize` command removes all data files, including customized configuration and log files, by unlinking the files from their directories. The command removes all user-created files from the system, including all plain-text passwords, secrets, private keys for SSH, local encryption, local authentication, IPsec, RADIUS, TACACS+, and SNMP.

This command also reboots the device and sets it to the factory default configuration. After the reboot, you cannot access the device through the management Ethernet interface. Log in through the console as `root` and start the Junos CLI by typing `cli` at the prompt as per the details in the initial configuration section.

Initial Configuration

When you power on a Junos device the first time, it will automatically boot and start. You must enter basic configuration information so that the device can communicate with other network devices and can be accessed over the network using either an in-band network management IP address, typically a loopback, or an out-of-band IP address, typically `fxp0` or `em0`.

Initially, to configure Junos you must connect a terminal or laptop computer into the console port. The console port may be located on the front or the rear of the device, depending on the equipment you are using. Console access to the device is enabled by default. Remote management access and all management access protocols, including Telnet, FTP, and SSH, are disabled by default for security reasons.

To configure Junos on the device for the first time follow along.

Connect a terminal or laptop to the console port and power on the Junos device. You can watch the boot up sequence from the terminal or laptop if you desire. When the device has booted you will be presented with the `root@#` prompt. Type in `cli`:

```
root@# cli
root@>
```

Enter Junos configuration mode by typing in `configure`:

```
cli> configure
[edit]
root@#
```

You are now ready to begin configuring the device.

First, let's configure the device with a hostname that is meaningful. I recommend standardizing hostnames across your estate for consistency. If you require spaces in the hostname enclose the entire name in quotation marks (" ").

The hostname is the name that identifies the device on the network and is easier to remember than an IP address. The command is:

```
[edit]
root@# set system host-name hostname
```

Next, configure the IP address and prefix length for the device management Ethernet interface. The management Ethernet interface provides a separate out-of-band management network for the router.

```
root@# set interfaces fxp0 unit 0 family inet address address/prefix-length
```

On devices that use management Ethernet interface em0, use it in place of fxp0.

Alternatively, you can configure a loopback address to manage the device from within your network. This will require a routing protocol to advertise the loopback address or static routes applying elsewhere in the network:

```
root@# set interfaces loopback0 unit 0 family inet address address/prefix-length
```

Next I recommend configuring the system services. This is how you will access the device remotely using DHCP, Finger, FTP, rlogin, SSH, Telnet, and so on. Remote access to a Junos device is disabled by default for security reasons. You must explicitly configure how the device can be accessed; for instance, to configure the device to be accessed remotely using SSH:

```
root@# set system services ssh
```

Include `protocol-version` if you wish to specify the version of SSH:

```
root@# set system services ssh protocol-version version
```

The device will need at least one interface configured to communicate with other network devices. You can configure physical interfaces as well as the logical interfaces on your device. Interface types and properties will be covered later on in this chapter.

Configure an IP address for a backup router. It's called the backup router because it is used only while the routing protocol process is not running. Choose a device that is directly connected to the local device by way of the management interface. The device uses this backup router only when it is booting or when the Junos routing software (the routing protocol process, RPD) is not running:

```
root@# set system backup-router address
```

Configure the IP address of a DNS server. The router uses the DNS name server to translate hostnames into IP addresses:

```
root@# set system name-server address
```

Now set the root password, entering either a clear-text password that the system will encrypt, a password that is already encrypted, or an SSH public key string:

```
root@# set system root-authentication plain-text-password
New password: type password
Retype new password: retype password
```

To enter a password that is already encrypted, use the following command:

```
root@# set system root-authentication encrypted-password encrypted-password
```

To enter an SSH public key, use the following command:

```
root@# set system root-authentication ssh-rsa key
```

Commit the configuration, which activates the configuration on the device and test connectivity:

```
root@# commit
```

After committing the configuration, you'll see the newly configured hostname appear after the username in the prompt.

User Accounts

User accounts provide a way for users to access the device using on-box authentication. Alternatively, users can access devices without accounts if you configure RADIUS or TACACS+ servers.

For each account, you define the login name for the user and, optionally, information that identifies the user. After you have created an account, the software creates a home directory for the user.

For each user account, you can define the following:

- Username: Name that identifies the user. It must be unique within the device. Do not include spaces, colons, or commas in the username. The username can be up to 64 characters long.
- User's full name: (Optional) If the full name contains spaces, enclose it in quotation marks. Do not include colons or commas.
- User identifier (UID): (Optional) Numeric identifier that is associated with the user account name. The identifier must be in the range from 100 through 64,000 and must be unique within the device. If you do not assign a UID to a username, the software assigns one when you commit the configuration, preferring the lowest available number.
- User's access privilege: (Required) One of the login classes you defined in the `class` statement at the `[edit system login]` hierarchy level, or one of the default classes.

So, let's go ahead and configure a new user called *noc-user* with operator login class privileges. Login classes will be covered in the very next section. If you configure the `plain-text-password` option, you are prompted to enter and confirm the password:

```
user@host# set system login user noc-user class operator authentication plain-text-password
New password: type password here
Retype new password: retype password here
```

Login Classes

Login classes are a way to define access privileges, permission for using CLI commands and statements, and session idle time for each login account. You can apply a login class to an individual user account, thereby specifying certain privileges and permissions to each user if desired.

There are four predefined classes within the Junos operating system:

- Operator: Can clear, network, reset, trace, and view
- Read-only: View
- Superuser or super-user: All
- Unauthorized: None

Be careful not to grant super-user access to anyone who isn't extremely familiar with Junos. This gives a user ultimate reign over the operating system.

Let's create a custom built login class to give you some idea of how this works. I'm going to create a login class for a NOC user called *noc-user-class*. For a list of available permissions use a question mark (?) to see what options exist:

```
user@host# set system login class noc-user-class permissions ?

Possible completions:
  [                    Open a set of values
  access               Can view access configuration
  access-control       Can modify access configuration
  admin                Can view user accounts
  admin-control        Can modify user accounts
  all                  All permission bits turned on
  clear                Can clear learned network info
  configure            Can enter configuration mode
  control              Can modify any config
  field                Can use field debug commands
  firewall             Can view firewall configuration
  firewall-control     Can modify firewall configuration
  floppy               Can read and write the floppy
  flow-tap             Can view flow-tap configuration
  flow-tap-control     Can modify flow-tap configuration
  flow-tap-operation   Can tap flows
  idp-profiler-operation   Can Profiler data
  interface            Can view interface configuration
  interface-control    Can modify interface configuration
  maintenance          Can become the super-user
  network              Can access the network
  pgcp-session-mirroring   Can view pgcp session mirroring configuration
  pgcp-session-mirroring-controlCan modify pgcp session mirroring configuration
  reset                Can reset/restart interfaces and daemons
  rollback             Can rollback to previous configurations
  routing              Can view routing configuration
  routing-control      Can modify routing configuration
  secret               Can view secret statements
```

```
  secret-control          Can modify secret statements
  security                Can view security configuration
  security-control        Can modify security configuration
  shell                   Can start a local shell
  snmp                    Can view SNMP configuration
  snmp-control            Can modify SNMP configuration
  storage                 Can view fibre channel storage protocol configuration
  storage-control         Can modify fibre channel storage protocol configuration
  system                  Can view system configuration
  system-control          Can modify system configuration
  trace                   Can view trace file settings
  trace-control           Can modify trace file settings
  unified-edge            Can view unified edge configuration
  unified-edge-control      Can modify unified edge configuration
  view                    Can view current values and statistics
  view-configuration      Can view all configuration (not including secrets)
```

Permissions are known as permission bits. In this example I'd like the noc-user-class to be able to `clear` (delete) information learned from the network that is stored in various network databases (using the `clear` commands), `network` access the network by entering the `ping`, `ssh`, `telnet`, and `traceroute` commands, `trace` file settings in configuration and operational modes, `view` use various commands to display current system-wide, routing table, and protocol-specific values and statistics, `view-configuration` view all configuration (not including secrets).

So, how does this look in the CLI?

```
user@host# set system login class noc-user-class permissions [ clear network trace view view-
configuration ]
```

At this point I want to throw in two really useful added extras to the login class configuration. This is something I always add to every user I configure:

```
user@host# set system login class noc-user-class login-alarms
user@host# set system login class noc-user-class login-tip
```

The `login-alarms` show system alarms automatically when a user logs in; this is very helpful to spot any issues when first logging in. This is also a great way to remind yourself to create a rescue configuration as you will see when we test this user.

The `login-tip` command provides the option of configuring login tips for the user. By default, the `tip` command is not enabled when a user logs in. Again, this will be shown when testing this user's access.

Finally, I recommend configuring an idle-timeout. This is the maximum amount of time a user can be idle before the user is logged out of the system. The session will time out after remaining at the CLI operational mode prompt for the specified time:

```
user@host# set system login class noc-user-class idle-timeout idle-imeout
```

For completeness let's create a login user called *noc-user* and associate the user with login class *noc-user-class* so that we can test our configuration:

```
user@host# set system login user noc-user class noc-user authentication plain-text-password
New password: type password here
Retype new password: retype password here
```

Now commit the changes and log in using the new user to test the configuration. Now when I log in you will see a Junos tip and any current active alarms as per below:

```
JUNOS tip:
Use 'load patch' to load structured patch files that follow the
unified patch style. 'show | compare' generates this style.

3 alarms currently active
Alarm time               Class  Description
2019-11-25 16:00:21 UTC  Major  FPC Management1 Ethernet Link Down
2019-11-25 16:00:03 UTC  Minor  Rescue configuration is not set
```

If I try and enter configuration mode the device will respond with `unknown-command` as this user is not privileged to enter configuration mode.

```
user@host# configure
          ^
unknown command.
```

NOTE It is also possible to add `deny` commands using the `deny-commands` statement.

Configuration Archival

As you learned in Chapter 2, Junos saves up to 49 rollback configurations in the local file system. Does it mean that you still need to have a configuration backup on an external server? Definitely, yes. Remember that bad things sometimes happen, such as flash memory becoming corrupt or even network devices being stolen.

Junos allows you to easily configure automatic backup (archival) of configurations. This is how you can do it. First, enter configuration mode and change to the `[edit system archival configuration]` hierarchy:

```
lab@vMX-1> configure
Entering configuration mode

[edit]
lab@vMX-1# edit system archival configuration

[edit system archival configuration]
lab@vMX-1#
```

You have two options: you can configure the device to send configuration backup at every commit, or, based on the time period interval, specified in minutes. In the former case, you do it this way:

```
[edit system archival configuration]
lab@vMX-1# set transfer-on-commit
```

In the latter case, specify it as follows (in this case the interval is set to 60 minutes, or one hour):

```
[edit system archival configuration]
lab@vMX-1# set transfer-interval 60
```

It is important to know that you can't have both `transfer-interval` and `transfer-on-commit` in your configuration. If you try to configure both, only the option that was configured last will be used and present in the configuration.

Finally, you must specify `archive-sites` URL(s) that define where to send archived configs to. You can use FTP, HTTP, or SCP protocols to transfer files to the remote server (saving to a local storage is possible, but it does not make much sense). You will often also need to specify the password on the remote server. Here's a complete example for archiving configuration after every commit using SCP protocol connection:

```
[edit system archival configuration]
lab@vMX-1# show
transfer-on-commit;
archive-sites {
    scp://lab@10.254.0.42:/var/home/lab password $9$A6UytORhclvMXREdb2gJZ; ## SECRET-DATA
}
```

Now it only takes a commit for this device to send an archived configuration to the destination server (in this example, 10.254.0.42). To monitor and troubleshoot the Junos configuration archival, you have the following options.

System log file will contain messages similar to this one, assuming successful transfer:

```
May  2 19:08:15  vMX-1 logger: transfer-file: Transferred /var/transfer/config/vMX-
1_20200502_190744_juniper.conf.gz
```

In case of a failure, you will see something like this:

```
May  2 18:12:10  vMX-1 logger: transfer-file failed to transfer /var/transfer/config/vMX-
1_20200502_180612_juniper.conf.gz rc 1
```

The files that a Junos device sends to an archive URL are first cached locally in the `/var/transfer/config` directory. You should only see files in that directory for a short period of time. If they keep accumulating in that directory, there must be a problem transferring them.

The destination filename format is as follows:

```
host-name_YYYYMMDD_HHMMSS_juniper.conf.gz
```

with timestamp in UTC time zone. Here is an example archived config file list on the destination server (in this lab example another Junos device, vMX-2, is acting as a destination server):

```
lab@vMX-2> file list vMX*
/var/home/lab/vMX-1_20200502_180625_juniper.conf.gz
/var/home/lab/vMX-1_20200502_190744_juniper.conf.gz
/var/home/lab/vMX-1_20200502_190809_juniper.conf.gz
```

Here are some extra considerations to keep in mind in relation to the Junos configuration archival. If you specify multiple `archive-sites`, the device will attempt to transfer files to the first archive site URL in the list, moving to the next site only if the transfer fails. And, when you specify `archive-sites`, do *not* add a forward slash (`/`) to the end of the URLs.

The value of `transfer-interval` that you can configure is from 15 through 2880 minutes.

Logging and Tracing

Imagine trying to troubleshoot an issue without any information to help guide you, for example, you know there is a connectivity issue but no idea what's causing it. This is where logs come in and where the Junos OS will record events and errors in specific log files, and for when things get really serious, debugging is the next step in problem resolution. This section addresses the different log files and debugging, which Junos refers to as tracing.

Log Files

Let's look at logging first. If you look at your `/var/log/ path`, you will see all sorts of log files. The top three files that you should have a look at are:

- chassisd: Contains all kinds of information belonging to your chassis process.
- interactive-commands: Contains all commands that every user has used on the CLI (if configured).
- messages: Depending on the severity, it logs up to almost all messages.

Junos generates system log messages (also called *syslog messages*) to record *system events* that occur on the device. Events consist of operations, failure, and error conditions that might require a resolution. This system logging utility is similar to the UNIX *syslogd* utility.

Each Junos system log message belongs to a message category, called a *facility*, that reflects the hardware- or software-based source of the triggering event. A group of messages belonging to the same facility is either generated by the same software process or concerns a similar hardware condition or user activity (such as authentication attempts). Each system log message is also preassigned a *severity*, which indicates how seriously the triggering event affects router (or switch) functions. Together, the facility and severity of an event are known as the message

priority. The content of a syslog message identifies the Junos *process* that generates the message and briefly describes the error that occurred.

By default, syslog messages that have a severity of `info` or more serious are written to the main system log file messages in the /var/log directory of the local RE.

For all syslog facilities or for a specified facility, you can configure the syslog message utility to redirect messages of a specified severity to a specified file instead of to the main system log file. You can also configure the syslog message utility to write syslog messages of specified severity—for all syslog facilities or for a specified facility—to additional destinations as discussed earlier.

And you can override the default values for file-archiving properties and the default timestamp format for all system logging messages—regardless of facility, severity, or destination—at the global level.

When troubleshooting these three logfiles very quickly become your main sources. Depending on the type of severity, this can be extremely useful. You should also notice that some of these files will rotate on your file system, so it might be worth it to check every log file on the system and not just the latest.

If you want to get a live monitoring while looking at your console, you can use the `monitor start <filename>` command. As soon as a message is written to your monitored file, it will additionally be redirected to the console and shown live.

For example:

```
monitor start messages
```

To stop the live monitor simply use:

```
monitor stop messages
```

This is extremely handy if you are live troubleshooting and want to see every message on your device. But did you know that there is actually another way that you can enable your Junos device to see even more? It's called *traceoptions*.

Traceoptions

Traceoptions can be configured for almost every protocol that you run on your Junos device. It lets you quickly see what exactly is going wrong. However, in most cases, they really occupy your CPU – so you should always disable them when you stop troubleshooting. Traceoptions are a good tool to verify and to troubleshoot everything in Junos. Tracing operations will record more detailed messages about the operation of routing protocols, such as the various types of routing protocol packets sent and received, and routing policy actions.

For example, let's look at the traceoptions for OSPF:

```
set protocols ospf traceoptions file ospf-log
set protocols ospf traceoptions file files 5 size 10k
set protocols ospf traceoptions flag lsa-ack
set protocols ospf traceoptions flag database-description
set protocols ospf traceoptions flag hello
```

The first line declares the filename `ospf-log`. We also define the number of files and the size of files, so that our system will not have any disk space issue in the future. This is generally a good idea. It's advised to collect as much data as you need, but also as few as needed in order to pinpoint the issue and not get overwhelmed with lots and lots of logs. Some engineers even write their own syslog-parsers to quickly identify the needed parts out of hundreds of GB of logfiles.

Traceoptions use flags to set what you want to see. In our case, we want to see everything that belongs to either `lsa-ack`, `database-description`, or the `hellos`. Can we even see more? Yes, use the ? you see the full list of possible completions:

```
Possible completions:
  all                  Trace everything
  backup-spf           Trace backup SPF (LFA) specific events
  database-description  Trace database description packets
  error                Trace errored packets
  event                Trace OSPF state machine events
  flooding             Trace LSA flooding
  general              Trace general events
  graceful-restart     Trace graceful restart
  hello                Trace hello packets
  ldp-synchronization  Trace synchronization between OSPF and LDP
  lsa-ack              Trace LSA acknowledgment packets
  lsa-analysis         Trace LSA analysis
  lsa-request          Trace LSA request packets
  lsa-update           Trace LSA update packets
  normal               Trace normal events
  nsr-synchronization  Trace NSR synchronization events
  on-demand            Trace demand circuit extensions
  packet-dump          Dump the contents of selected packet types
  packets              Trace all OSPF packets
  policy               Trace policy processing
  post-convergence-lfa  Trace post-convergence-lfa related events
  restart-signaling    Trace restart signaling
  route                Trace routing information
  source-packet-routing  Trace source packet routing (SPRING) events
  spf                  Trace SPF calculations
  state                Trace state transitions
  task                 Trace routing protocol task processing
  timer                Trace routing protocol timer processing
```

This list might differ for every protocol, so that you can select very granularly what you want to see and what you don't want to see. Again – when troubleshooting, you should always try to pinpoint the issue – start big and get smaller, step by step. It is important to remember what traceoptions can do and where you can typically enable them. There are basically four ways you can configure them:

- Global tracing operations: tracing for all routing protocols. You define these tracing operations at the `[edit routing-options]` hierarchy level of the configuration.
- Protocol-specific tracing operations: Define tracing for a specific routing protocol. You define these tracing operations in the `[edit protocols]` hierarchy when configuring the individual routing protocol (as shown above for OSPF). Protocol-specific tracing operations override any equivalent operations that you specify in the global traceoptions statement. If there are no equivalent operations, they supplement the global tracing options. If you do not specify any protocol-specific tracing, the routing protocol inherits all the global tracing operations. It's the same rule as for all statements in Junos: more specific wins!
- Tracing operations within individual routing protocol entities: Some protocols allow you to define more granular tracing operations. For example, in Border Gateway Protocol (BGP), you can configure peer-specific tracing operations. These operations override any equivalent BGP-wide operations or, if there are no equivalents, supplement them. If you do not specify any peer-specific tracing operations, the peers inherit, first, all the BGP-wide tracing operations and, second, the global tracing operations.
- Interface tracing operations: Define tracing for individual router interfaces and for the interface process itself. You define these tracing operations at the `[edit interfaces]` hierarchy level of the configuration.

Rescue Configuration

Junos OS allows you to save a special version of configuration file, known as the rescue configuration. For the rescue, you normally save a version of configuration that is tested and known to work well. It only takes one command to do it:

```
lab@vMX-1> request system configuration rescue save
lab@vMX-1>
```

This command saves current *active* configuration as rescue configuration. As you see, there is no output for this command, but you can find the rescue config file in `/config` directory:

```
lab@vMX-1> file list /config/
/config/:
db_ext
juniper.conf.1.gz
juniper.conf.2.gz
juniper.conf.3.gz
juniper.conf.gz
juniper.conf.md5
license/
rescue.conf.gz
scripts/
```

```
usage.db
vchassis/
vmx_initialized
```

If things ever go wrong while configuring your device, to rollback to a known good (rescue) state of your config, just do `rollback rescue` and `commit` like this:

```
[edit]
lab@vMX-1# rollback rescue
load complete

[edit]
lab@vMX-1# commit
commit complete
```

Some Junos devices (such as the SRX300) also have a RESET CONFIG button. Pressing and quickly releasing the RESET CONFIG button loads and commits the rescue configuration, thus having the same effect as the above two commands. However, pressing and holding the RESET CONFIG button for 15 seconds or more deletes all configurations on the device, including the backup configurations and rescue configuration, and loads and commits the factory default configuration. The RESET CONFIG button can be disabled in configuration, if needed.

Finally, if you want to delete the rescue configuration, you can do it as follows (again, there is no output):

```
lab@vMX-1> request system configuration rescue delete
lab@vMX-1>
```

You can now check that the rescue config is no longer present:

```
lab@vMX-1> file list /config/rescue.conf.gz
/config/rescue.conf.gz: No such file or directory
```

By the way, if you are not having a rescue configuration set, Junos will warn you with a system alarm, like this:

```
lab@vMX-1> show system alarms
1 alarms currently active
Alarm time               Class  Description
2020-05-02 19:58:53 UTC  Minor  Rescue configuration is not set
```

Basic Components

Once the hostname and root password have been set in the initial configuration, the next step is to configure interfaces, add a domain name, and add DNS servers. Engineers may also want to remove some existing configuration entries, too. In short, engineers need to know how to configure some of the more basic components in Junos.

Let's look at a sample config of a vQFX (cutting the SSH key for better readability):

```
root@vqfx-re> show configuration | display set
set version 20191212.201431_builder.r1074901
set system host-name vqfx-re
set system root-authentication encrypted-password

set system root-authentication ssh-rsa "ssh-rsa

set system login user vagrant uid 2000
set system login user vagrant class super-user
set system login user vagrant authentication ssh-rsa "ssh-rsa

set system services ssh root-login allow
set system services netconf ssh
set system services rest http port 8080
set system services rest enable-explorer
set system syslog user * any emergency
set system syslog file messages any notice
set system syslog file messages authorization info
set system syslog file interactive-commands interactive-commands any
set system extensions providers juniper license-type juniper deployment-scope commercial
set system extensions providers chef license-type juniper deployment-scope commercial
set interfaces em0 unit 0 family inet dhcp
set interfaces em1 unit 0 family inet address 169.254.0.2/24
set forwarding-options storm-control-profiles default all
set protocols igmp-snooping vlan default
set vlans default vlan-id 1
```

As you can see, our Junos device has a hostname configured (QFX-2), has allowed the root user to SSH into the device (`root login allow`), uses the default syslog config, meaning that it logs any notice to the messages log file, has exactly one VLAN configured (`vlan-id 1` with the name default) and has no interfaces so far. In Junos you can have a valid minimalistic config containing only a single line: the root password. The root password is basically the only line that is mandatory. If you try to commit a config without a root password, the device will warn you and the commit will fail. In this case our config is missing a crucial part: DNS. Leaving the config like this will never allow our device to ping hostnames, for example, because a name server to translate the name into an IP is missing.

Let's change that:

```
{master:0}
root@QFX-2> configure
Entering configuration mode

{master:0}[edit]
root@QFX-2# set system name-server 8.8.8.8

{master:0}[edit]
root@QFX-2# show | compare
[edit system]
+  name-server {
+      8.8.8.8;
+  }
```

```
{master:0}[edit]
root@QFX-2# commit
configuration check succeeds
commit complete

{master:0}[edit]
root@QFX-2#
```

As you can see, we have successfully added 8.8.8.8 (Google DNS) as our DNS server. Many organizations run their own DNS servers so that your internal devices do not necessarily have to reach the internet to get DNS-name-resolution – but that depends entirely on your organization. Our QFX now has a hostname, password, and DNS but is still missing something: interfaces.

Although technically two interfaces are assigned (em0 and em1), we can't reach our device because em0 and em1 are management interfaces only. So let's add some usable interfaces. Our Router-A is connected via 10.10.10.10/24 over interface 12 via 10Gigabit. If you encounter something like this you need to remember, that Junos does *not* start counting from 1 but from 0! So if your colleague has never seen Junos and starts counting the port numbers and tells you it's connected to port 12 you should be aware of the fact, that he might mean interface 11, since 0 is also an interface. In our case, we assume he knows Junos and really means interface 12. 10Gigabit interfaces are xe-Interfaces so this would be xe-0/0/12. Brilliant – we got our interface. Let's add an IP to it:

```
set interfaces xe-0/0/12 unit 0 family inet address 10.10.10.10/24
set interfaces em0 unit 0 family inet dhcp
set interfaces em1 unit 0 family inet address 169.254.0.2/24
```

As you can see, we assigned the unit to our interface (as discussed earlier) and also used the family inet for a Layer 3 (IP) interface with the IP of 10.10.10.10 with the netmask 255.255.255.0 (/24). If we commit this configuration and connect our Router-A to port 12 of the QFX, we will have the possibility to SSH from Router-A over 10.10.10.10 to the QFX with the root-user. The QFX is now able to use DNS and NTP (NTP itself will be covered in a later chapter) and is now ready to be configured further (with SNMP for example) to query the QFX device over xe-0/0/12 and get some nice graphs. Don't worry – we will do that in a moment.

User Authentication Methods

To get access to your Junos devices, Junos supports different methods such as local password authentication (often referred to as local database), RADIUS, and TACACS+, to control access to the device itself or the network. Authentication methods are generally used for validating users who attempt to get access to the router or switch using protocols like SSH or TELNET. Authentication prevents unauthorized devices and users from gaining access and is mandatory when starting to configure a Junos device.

Junos supports three methods of user authentication:

- Local password authentication (local database)
- Remote Authentication Dial-In User Service (RADIUS)
- Terminal Access Controller Access Control System Plus (TACACS+)

Let's cover these three methods in detail so that you know precisely what to configure when.

Local Password Authentication (local database)

With local password authentication, you configure a password for each user allowed to log in to the Junos device. As you can imagine, you have to continually make sure that all of your devices are always up-to-date in terms of your configured users and passwords. This can be problematic in scenarios where the net admin is leaving the company. In this case, you would need to connect to all your devices in your entire infrastructure and remove his / her user or change the password. Missing one device in this way can be fatal in the future, especially if you think about audits and pentesting. Although local password authentication can be increased by using SSH-keys instead of plaintext passwords, this method is often used in smaller networks with few administrators. In general, you should set a very strong root-password and create a user for yourself so that you can lock the root password somewhere safe (for example, during maintenance) where only a few people have access to it.

CAUTION The local password hashing can change from version to version. Keep that in mind when downgrading your device – because if you set a password in a higher version and then issue a downgrade, the lower version might not recognize your password anymore, and you need to proceed with the root-recovery procedure!

- In Junos OS Release 12.3 and earlier, MD5 encryption is used, and the password starts with 1.
- In Junos OS release 15.1, SHA-256 encryption is used, and the password starts with 5.
- Starting with Junos 17.2, SHA-512 encryption is used, and the password starts with 6.

RADIUS and TACACS+

RADIUS and TACACS+ are both distributed client-server systems—while RADIUS and TACACS+ clients run on your Junos device, the server runs on a remote server and your device needs to be able to talk to it. This is important because if the connection fails, the system can no longer verify your credentials, and depending on the configuration might even lock you out completely. You can configure the router or switch to be both a RADIUS and TACACS+ client, and you can also configure the local password authentication in the Junos configuration file. This way it is possible to prioritize the methods and set the order in which the software tries the different authentication methods when verifying user access. So if your RADIUS fails, you can tell your Junos to use the local password as a last resort.

To make this more comfortable for you, I've included some examples from Juniper's expert documentation in Table 3.1.

Table 3.1: Order of Authentication Attempts

Syntax	Order of Authentication Attempts
authentication-order radius;	Try configured RADIUS authentication servers.
	If RADIUS server is available and authentication is accepted, grant access.
	If RADIUS server is available but authentication is rejected, deny access.
	If RADIUS servers are not available, try password authentication.
	Note: If a RADIUS server is available, password authentication is not attempted, because it is not explicitly configured in the authentication order.
authentication-order [radius password];	Try configured RADIUS authentication servers.
	If RADIUS servers fail to respond or return a reject response, try password authentication, because it is explicitly configured in the authentication order.
authentication-order [radius tacplus];	Try configured RADIUS authentication servers.
	If RADIUS server is available and authentication is accepted, grant access.

	If RADIUS servers fail to respond or return a reject response, try configured TACACS+ servers.
	If TACACS+ server is available and authentication is accepted, grant access.
	If TACACS+ server is available but authentication is rejected, deny access.
	If both RADIUS and TACACS+ servers are not available, try password authentication.
	Note: If either RADIUS or TACACS+ servers are available, password authentication is not attempted, because it is not explicitly configured in the authentication order.

As you can see, the order can be critical and is something that you need to think about carefully. There are some other things that you should also think about. Does your organization already use TACACS+? If yes, then Junos is ready for that. If you are not using it, you might, however, stick with RADIUS because RADIUS servers are a bit more lightweight, and the RADIUS protocol itself is a multivendor IETF standard, and its features are more widely accepted than those of TACACS+, or other proprietary systems. Both support IPv4 and IPv6.

The RADIUS configuration on Juniper is actually quite simple; however, to get Junos to authenticate a user and provide that user with specific privileges requires a little bit more configuration. You must define the RADIUS server using:

```
set system radius-server address 172.16.2.200
```

After defining your RADIUS server, you must set your so-called RADIUS secret –basically a phrase / password, that needs to match between your OS device and the RADIUS server. This is done via:

```
set system radius-server 172.16.2.200 secret mysupersecretradiussecret
```

If your device has multiple IPs configured, it is crucial to be able to set a source interface (IP) from which the RADIUS requests will be sent. Your RADIUS server is configured to listen to only a specific management address for the previously defined secret, so you want to make sure that you select the correct interface (with the correct IP) as your Junos source interface. Because RADIUS verifies the authenticating device by a secret passphrase, the IP address and passphrase must match. If a match is not found in the RADIUS database, your server will just drop the access-request. Think carefully about the correct interface. Often, a loopback interface is used so that RADIUS will still work if a particular interface goes down.

To set the Junos RADIUS source interface, we use:

```
set system radius-server 172.16.2.200 source-address 192.168.0.254
```

In this example, 192.168.0.254 is the IP address of the interface that sources all RADIUS traffic originated by the device.

Once you have configured the basic RADIUS server parameters, you must then configure a local default remote user (for example, `remote`). This user will be the default template for all RADIUS authenticated sessions. It is always a good practice to provide only minimal security access by default – read-only. To configure the default remote user, use:

```
set login user remote full-name RADIUS Template User uid 1337 class read-only
```

Finally, you must define the needed remote user classes to be used with the RADIUS responses from your RADIUS server. Simply create users with no passwords and assign them the correct corresponding permissions:

```
set system login user Callagents class read-only
set system login user Noc-operators class operator
set system login user Administrator class super-user
```

On your RADIUS server, the attribute you'll be using to respond to the authenticated request is called *Juniper-Local-User-Name*. The value for this attribute must be the corresponding user template defined on the Junos device. If you want your AD User called Christian.Scholz to be a super-user, you assign him the Juniper-Local-User-Name=Administrator value. I leave it up to you to make me an Administrator on all your devices. :D

If RADIUS is configured, the default authentication order is RADIUS first, then the local user accounts – but this happens *only* if the RADIUS server is unreachable! If you need local authentication first and RADIUS followed after, you can change the order to `[password radius]` or `[password radius tacplus]`.

Interface Types and Properties

Junos offers different interfaces depending on the type of media used (Ethernet, logical, dial-in, etc.) Table 3.2 lists the interfaces that you might encounter in the wild.

Table 3.2 Interface Types

Console port (con)	Almost every Junos Device has a serial port labeled CON or CONSOLE, for connecting tty-type terminals to the device using standard PC-type tty cables. The console port does not have a physical address or IP address associated with it.
Loopback (lo0, lo1…)	All Junos devices have this software-only virtual interface that is always up. The loopback interface provides a stable and consistent interface and IP address on the switch.

Management interface (me, vme, em, fxp…)	Junos uses management Interfaces called me0, vme, or em0, depending on the device you use. The management Ethernet interface provides an out-of-band method for connecting to the switch. To use the port as a management port, you must configure its logical port (for example me0.0), with a valid IP address.
Integrated Routing and Bridging (irb) Interface or Routed VLAN Interface (rvi)	Your Junos devices use an integrated routing and bridging (IRB) interface or Routed VLAN Interface (RVI) to route traffic from one broadcast domain to another and to perform other Layer 3 functions such as traffic engineering. The IRB interface or RVI functions as a logical router, eliminating the need for having both a switch and a router. These interfaces must be configured as part of a broadcast domain or virtual private LAN service (VPLS) routing instance for Layer 3 traffic to be routed from.
Virtual Chassis port (VCP) interfaces	Virtual Chassis ports (VCPs) are used to interconnect switches in a Virtual Chassis.
Multichassis aggregated Ethernet (MC-AE) interfaces	Group a LAG on one standalone switch with a LAG on another standalone switch to create an MC-AE. The MC-AE provides load balancing and redundancy across the two standalone switches.

Many hardware interfaces can be combined with logical interfaces and subinterfaces. The `unit` is the designation that Junos uses to allow multiple logical interfaces to be configured on a single physical interface. The interface is for physical configurations and the unit is used for logical configuration parameters (like it's IP). An example would be:

```
set interfaces ge-0/0/0.0 family inet address 172.16.2.1/24
set interfaces ge-0/0/0 unit 1 family inet address 172.16.2.1/24
```

While you only have one physical interface, ge-0/0/0, it uses the subinterfaces (units) either by writing out the unit or using a dot `.` to signal, that this is a subinterface. Rather than adding a unit designation for those interfaces that have multiple logical interfaces, Junos requires the unit designation on all interfaces with a default first unit number of zero. This requirement makes all interface names consistent regardless of whether or not there are multiple logical interfaces.

Depending on the interface type, the name of the interface changes, but follows the schema:
type-fpc/pic/port

The most common types that you will face are listed in Table 3.3.

Table 3.3 Common Junos Interfaces

ae	Aggregated Ethernet interface. This is a virtual aggregated link and has one or more member links associated with it.
es	Encryption interface
fxp	Management and internal Ethernet interfaces
gr	Generic routing encapsulation (GRE) tunnel interface
lo	Loopback interface. The Junos OS automatically configures one loopback interface (lo0).
so	SONET/SDH interface
vt	Virtual loopback tunnel interface
fe	Fast Ethernet interface (100m)
ge	Gigabit Ethernet interface (1G)
xe	10-Gigabit Ethernet interface
et	40, and 100-Gigabit Ethernet

MORE? A full list can be found at: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/router-interfaces-overview.html#id-interface-naming-overview.

Configuration Groups

The configuration groups feature in Junos enable you to create a group containing configuration statements and to direct the inheritance of that group's statements in the rest of the configuration. The same group can be applied to different sections of the configuration, and various sections of one group's configuration statements can be inherited in different places in the configuration.

To see this, let's look at an example:

```
root@Junos> show groups mygroup

interfaces {
    <ge-*> {
        speed 1g;
        link-mode full-duplex;
        }

    }
}
```

After creating this group, you need to activate it:

```
set interfaces apply-groups mygroup
```

If you look at the interfaces with the `inheritance` CLI parameter, which tells Junos to give us the configuration including all the hierarchy, you can see that your interface, ge-0/0/3, got some info from the `mygroup` config statement:

```
root@Junos> show interfaces ge-0/0/3 | display inheritance

description I did not configure speed or link mode here;
##
## '1g' was inherited from group ' mygroup '
##
speed 1g;
##
## 'full-duplex' was inherited from group ' mygroup '
##
link-mode full-duplex;
```

Configuration groups enable you to create smaller, more logically constructed configuration files, making it easier to configure and maintain Junos and making your configuration a lot easier to understand. You can group statements that are repeated in many places in the configuration, such as when configuring interfaces, and thereby limit updates to only the group.

You can also use wildcards in a configuration group to allow configuration data to be inherited by any object that matches a wildcard expression (as seen above).

The configuration group mechanism is separate from the grouping mechanisms used elsewhere in the configuration, such as BGP groups. Configuration groups provide a universal tool that can be used throughout the configuration, but that is known only to the Junos OS CLI. The individual software processes that perform the actions directed by the configuration receive the expanded form of the configuration; they have no knowledge of configuration groups.

Configuration groups use true inheritance, which involves a dynamic, ongoing relationship between the source of the configuration data and the target of that data. Data values changed in the configuration group are automatically inherited by the target. The target does not need to contain the inherited information, although the inherited values can be overridden in the target without affecting the source from which they were inherited.

In Junos there is always a rule: more specific wins. If you configure the speed via a group statement, but also configure a colliding speed at the interface level, your interface level is more specific, and the group statement for this interface will be overwritten by the more specific one under your interface. Keep that in mind because this also applies to firewall policies, parameters inside your configuration, or even IP-Addresses.

NTP

"The Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks." Thanks, Wikipedia – but what does that mean for our Junos devices? It's simple: imagine having five switches, and each has a different time. Your colleague calls you and reports that a failure occurred. You now fetch all the logs from all five devices to see what's wrong. Oh no! Switch 1 had the issue arising at 12.13, I think, or was it 15.20? Or 20.45? We don't know for sure, because we don't have the same persistent time across our devices, making it impossible to correctly troubleshoot. Therefore, we can get our Junos devices to synchronize the time with an NTP Server. The good part is that this clock does not even necessarily have to be correct – but it's consistent – and that's what we need. If all our devices have the wrong time but all of them use the exact same wrong time, you can still troubleshoot the issue very precisely because they all agreed at the same time. Let's see how to do this. For NTP, there are three commands to remember:

```
set ntp boot-server 10.1.4.1
set ntp server 10.1.4.2
set ntp authentication-key 2 type md5 value $ABC123
```

The first tells your Junos device the IP of your NTP server at boot-time. Why? Simple: if your time on the NTP server and the time on your device differ too much, they will not sync anymore. The `boot-server` parameter tells the switch to just take the time, regardless of the difference at boot-time, and use it. After that, the second parameter kicks in (and they can even have different IPs as shown) and tells the device to establish a session with the given IP to permanently monitor and synchronize its local time. If you don't want external devices to be able to talk to your NTP server, you can also configure an *md5 authentication*. Now all your devices have the same time, and this will help you tremendously.

These reference time sources are very precise atomic clocks and called *Stratum 0* in NTP terminology. Each server from there increases the Stratum, and each server getting the time from that server further increases it, and so on. Most of the time network devices have less accurate clocks, which may deviate from these reference clocks after some period of time, even though you set it correctly.

SNMP

Flashy graphics, nice diagrams – we all love statistics and data. Sooner or later, you will have the need to permanently monitor your Junos device. While you could use telemetry, telemetry is still expensive most of the time and requires specific hardware, so there's a trusty old protocol that you can use: SNMP (Simple

Network Management Protocol). SNMP lets you very quickly check how your device is behaving (fan speed, temperature) or let you read statistics (errors, counters, traffic, etc.) as shown in Figure 3.1.

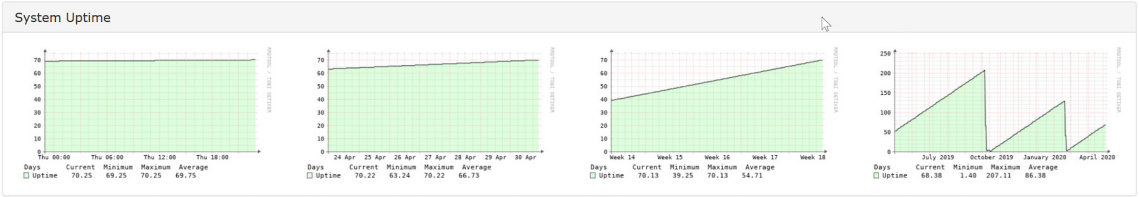


Figure 3.1 SNMP

Let's have a look at how to configure it:

```
set snmp location Lab 4 Row 11 contact contact@mail.tld
set snmp community public authorization read-only
set snmp community public client-list-name list0
set snmp client-list list0 192.168.0.0/24

set snmp trap-group qf-traps destination-port 155 targets 192.168.0.100
```

The location lets you set a position (who would have guessed, right?), where your device is placed (for example, in the lab). Your monitoring system will show you this location. Some NMS may even allow you to set a geo-location on a map. SNMP uses passwords, so-called *communities* to allow the query to itself. This is true for SNMPv1 and v2. Meanwhile, SNMP has reached Version 3 –allowing you to configure usernames and passwords to further increase security. In this case, the community name *public* is used. So in your NMS, you have to use public as well when discovering the Junos Device – or else the discovery will fail because the communities do not match. If you need to use SNMPv2, because your NMS is not capable of SNMPv3, you can increase the security with the `access-list` command. It contains a name (for example list0) and allowed IPs – only these IPs may query your Junos Device. SNMP uses a query system, so your NMS will fetch the information from time to time.

In some cases, for example, if a link goes down you want the switch to make your NMS aware of that before the next discovery. That's what the *traps* are for. Traps are SNMP messages triggered by the Junos device into the direction of your NMS to tell it that a situation has changed. In Junos, you configure the traps by configuring a `trap-group`, a destination port, and the target, where the traps should be sent to. You can also set multiple destinations(targets).

Syslog

Logging to a Server

Junos allows you to send log messages either to a local file or user or to a remote system (Syslog Server) or all of them. To tell Junos to send syslog information to a server, you can add the following config:

```
set system syslog host 172.16.2.200 any any
```

This configuration command will make the device send syslog messages for any facility and any severity. A *facility* is the source of the message; a *severity* tells the Syslog, what infos, and how many infos to log (any or only some).

Instead of inserting any, you can choose whatever you think is interesting:

```
{master:0}[edit]
root@QFX-2# set system syslog host 172.16.2.200 ?
Possible completions:
  allow-duplicates     Do not suppress the repeated message
  any                  All facilities
+ apply-groups         Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
  authorization        Authorization system
  change-log           Configuration change log
  conflict-log         Configuration conflict log
  daemon               Various system processes
  dfc                  Dynamic flow capture
  exclude-hostname     Exclude hostname field in messages
  explicit-priority    Include priority and facility in messages
  external             Local external applications
  facility-override    Alternate facility for logging to remote host
  firewall             Firewall filtering system
  ftp                  FTP process
  interactive-commands  Commands executed by the UI
  kernel               Kernel
  log-prefix           Prefix for all logging to this host
  match                Regular expression for lines to be logged
+ match-strings        Matching string(s) for lines to be logged
  ntp                  NTP process
  pfe                  Packet Forwarding Engine
  port                 Port number
  routing-instance     Routing instance
  security             Security related
  source-address       Use specified address as source address
> structured-data      Log system message in structured format
  user                 User processes
```

Logging to a File

Instead of logging to a server, Junos also allows you (and does this per default) to log messages to a file:

```
set system syslog file messages any notice
set system syslog file messages authorization notice
```

These configuration statements will send messages from any facility with the notice severity to the file called *messages*. To prevent your device from running out of space, Junos uses a default that varies from device to device. An MX, for example, will write messages to a file until it reaches 1MB. You can tell Junos how big your files can get and also tell it how many files it should keep before rotating. This can be done via:

```
set system syslog archive size 5m
set system syslog archive files 5
```

Since size and a maximum are configured, these values will take precedence over the default behavior of the device.

To check this syslog file, you can use the `show log messages` command. With the `?` after show log you can see all log files on your device. On the file system level this is the /var/log/ folder:

```
Possible completions:
  <[Enter]>            Execute this command
  <filename>           Name of log file
  __policy_names_rpdc__  Size: 98, Last changed: Jun 25 17:50:40
  __policy_names_rpdn__  Size: 154, Last changed: Jun 25 17:50:41
  authd_libstats       Size: 0, Last changed: Jan 20 08:41:20
  authd_profilelib     Size: 0, Last changed: Jan 20 08:41:20
  authd_sdb.log        Size: 0, Last changed: Jan 20 08:41:20
  chassisd             Size: 553497, Last changed: Jun 25 17:50:41
  chassisd_snmp        Size: 609, Last changed: Jun 25 17:50:39
  cosd                 Size: 33693, Last changed: Jun 25 17:50:41
  coslib_l2cpd         Size: 0, Last changed: Jan 20 08:41:27
  dcd                  Size: 75598, Last changed: Jun 25 17:50:41
  dcd_commit_check     Size: 1871, Last changed: Jun 25 17:50:40
  default-log-messages  Size: 0, Last changed: Jun 25 17:43:29
  dfwc                 Size: 0, Last changed: Jan 20 08:41:17
  eccd                 Size: 392, Last changed: Jun 25 17:50:39
  ext/                 Last changed: Jan 20 08:41:25
  flowc/               Last changed: Jan 20 08:41:25
  ggsn/                Last changed: Jan 20 08:41:25
  gres-test-point      Size: 3858, Last changed: Jun 25 17:44:27
  interactive-commands  Size: 95765, Last changed: Jun 25 17:54:16
  inventory            Size: 609, Last changed: Jun 25 17:50:39
  jam_chassisd         Size: 0, Last changed: Jan 20 08:41:16
  jam_clksyncd         Size: 0, Last changed: Jan 20 08:41:20
  jam_cosd             Size: 0, Last changed: Jan 20 08:41:19
  jam_dcd              Size: 0, Last changed: Jan 20 08:41:17
  jam_dfwd             Size: 0, Last changed: Jan 20 08:41:17
  jam_l2ald            Size: 0, Last changed: Jan 20 08:41:28
  jdhcpd_era_discover.log  Size: 859, Last changed: Jun 25 17:44:01
  jdhcpd_era_discover.log.0  Size: 859, Last changed: Jan 20 08:41:32
  jdhcpd_era_discover.log.1  Size: 0, Last changed: Jan 20 08:41:30
  jdhcpd_era_solicit.log  Size: 858, Last changed: Jun 25 17:44:01
  jdhcpd_era_solicit.log.0  Size: 858, Last changed: Jan 20 08:41:32
```

```
  jdhcpd_era_solicit.log.1  Size: 0, Last changed: Jan 20 08:41:30
  jdhcpd_era_v4_blq.log  Size: 881, Last changed: Jun 25 17:44:01
  jdhcpd_era_v4_blq.log.0  Size: 881, Last changed: Jan 20 08:41:32
  jdhcpd_era_v4_blq.log.1  Size: 0, Last changed: Jan 20 08:41:30
  jdhcpd_era_v6_blq.log  Size: 881, Last changed: Jun 25 17:44:01
  jdhcpd_era_v6_blq.log.0  Size: 881, Last changed: Jan 20 08:41:32
  jdhcpd_era_v6_blq.log.1  Size: 0, Last changed: Jan 20 08:41:30
  jdhcpd_etl.log       Size: 0, Last changed: Jan 20 08:41:19
  jdhcpd_sdb.log       Size: 0, Last changed: Jan 20 08:41:30
  license              Size: 0, Last changed: Jan 20 08:41:29
  license_subs_trace.log  Size: 0, Last changed: Jan 20 08:41:28
  mastership           Size: 1101, Last changed: Jun 25 17:50:39
  messages             Size: 100906, Last changed: Jun 25 17:54:28
  mgd-api              Size: 0, Last changed: Jan 20 08:41:29
  na-grpcd             Size: 23447, Last changed: Jun 25 17:44:27
  pfe_flow_stats.txt   Size: 0, Last changed: Jan 20 08:41:28
  pfed_jdhcpd_trace.log  Size: 0, Last changed: Jan 20 08:41:30
  pm_longterm.txt      Size: 253448, Last changed: Jan 20 08:41:20
  sdk-vmmd             Size: 0, Last changed: Jan 20 08:41:25
  shmlog/              Last changed: Jan 20 08:41:17
  user                 Show recent user logins
  wtmp                 Size: 2308, Last changed: Jun 25 17:45:00
  wtmp.0.gz            Size: 108, Last changed: Jun 25 17:43:36
  wtmp.1.gz            Size: 85, Last changed: Jan 20 08:44:35
  |                    Pipe through a command
```

If you want to get a live monitoring while looking at your console, you can use the `monitor start <filename>` command. As soon as a message is written to your monitored file, it will additionally be redirected to the console and shown live.

For example:

```
monitor start messages
```

To stop the live monitor you simply use:

```
monitor stop messages
```

Chapter 4

Operational Monitoring and Maintenance

by Jeff Fry

The Junos CLI is traditionally the most common way to monitor, support, and troubleshoot your network device. You accomplish this by using the `show` and `monitor` commands. Alternatives include using J-Web, SNMP, physical LEDs, and third party tools. This chapter focuses on the CLI commands.

CLI Management

One of the first things you might want to know about the CLI when you are new to Junos are basics like: *Where are my files saved? Is there a user login timeout? What are my permissions when I log in?* These questions can all be answered easily with some simple CLI commands in operational mode.

First look at the basic `show cli` command:

```
user_1@mx960> show cli
CLI complete-on-space set to on
CLI idle-timeout set to 10080 minutes
CLI restart-on-upgrade set to on
CLI screen-length set to 24
CLI screen-width set to 80
CLI terminal is 'xterm'
CLI is operating in enhanced mode
CLI timestamp disabled
CLI working directory is '/var/home/admin_full'
```

You can see there is a timeout and it is a very large number. By default, when you leave your session open it remains idle for a few days. Probably not a good idea.

This could be changed with the `set cli idle-timeout <minutes>` operational mode command. The value can be configured between 0 and 100,000 minutes, with 0 meaning no timeout.

Any value that configures with the `set cli idle-timeout` in operational mode will take effect over your current session. If you want to set the value permanently, and make the value applicable to all users when they log in, or to a group of users, you can configure the timer within configuration mode in two different ways:

You can do it globally (for all users) under edit system:

```
{master:0}[edit system login]
user_1@QFX5110-LS1# set idle-timeout ?
Possible completions:
  <idle-timeout>       Maximum idle time before logout (1..60 minutes)
```

Or you can do it within a class, and session from any user assigned to that class with be affected by that value:

```
{master:0}[edit system login class superuser-local]
user_1@QFX5110-LS1# set idle-timeout ?
Possible completions:
  <idle-timeout>       Maximum idle time before logout (minutes)

{master:0}[edit system login]
user_1@QFX5110-LS1# set user user_1 class superuser-local
```

If you want to know what you are allowed to do after you log in, you can enter the `show cli authorization` command. The example below shows you user_1, who belongs to class `superuser-local`, and has all the permissions listed. This user can look at the configuration (view-configuration) and can also enter configuration mode to modify it (permissions configure):

```
user_1@QFX5110-LS1> show cli authorization
Current user: 'admin_full' login: 'user_1' class 'superuser-local'
Permissions:
    access      -- Can view access configuration
    access-control-- Can modify access configuration
    admin       -- Can view user accounts
    admin-control-- Can modify user accounts
    clear       -- Can clear learned network info
    control     -- Can modify any config
    edit        -- Can edit full files
    field       -- Can use field debug commands
    firewall    -- Can view firewall configuration
    firewall-control-- Can modify firewall configuration
    floppy      -- Can read and write the floppy
    interface   -- Can view interface configuration
    interface-control-- Can modify interface configuration
    maintenance -- Can become the super-user
    network     -- Can access the network
    reset       -- Can reset/restart interfaces and daemons
    rollback    -- Can rollback to previous configurations
    routing     -- Can view routing configuration
    routing-control-- Can modify routing configuration
```

```
    secret      -- Can view secret statements
    secret-control-- Can modify secret statements
    security    -- Can view security configuration
    security-control-- Can modify security configuration
    shell       -- Can start a local shell
    snmp        -- Can view SNMP configuration
    snmp-control-- Can modify SNMP configuration
    storage     -- Can view fibre channel storage protocol configuration
    storage-control-- Can modify fibre channel storage protocol configuration
    system      -- Can view system configuration
    system-control-- Can modify system configuration
    trace       -- Can view trace file settings
    trace-control-- Can modify trace file settings
    view        -- Can view current values and statistics
    view-configuration-- Can view all configuration (not including secrets)
    all-control -- Can modify any configuration
    flow-tap    -- Can view flow-tap configuration
    flow-tap-control-- Can modify flow-tap configuration
    flow-tap-operation-- Can tap flows
    idp-profiler-operation-- Can Profiler data
    pgcp-session-mirroring-- Can view pgcp session mirroring configuration
    pgcp-session-mirroring-control-- Can modify pgcp session mirroring configuration
    unified-edge-- Can view unified edge configuration
    unified-edge-control-- Can modify unified edge configuration
Individual command authorization:
    Allow regular expression: none
    Deny regular expression: none
    Allow configuration regular expression: none
    Deny configuration regular expression: none
```

After you enter configuration mode you can save the configuration file by simple typing `save filename`. The file goes to your home directory by default. If you are using RADIUS to authenticate, however, even if you are logging in with your username you might not have a local account configured on the router, but might be assigned to a local user that is shared with other users. You might be expecting your file to be going to `/var/home/user_1`, but in fact it is going somewhere else. When you checked with the `show cli` command, you probably noticed that the directory is included, but you can also quickly type `show cli directory`. Knowing this might save you a lot of confusion:

```
user_1@mx960> show cli directory
Current directory: /var/home/full
```

Another CLI trick that is extremely useful is the `show cli history`. You'll see the value of it soon as you look at this next example:

```
user_1@mx960> show cli history
  15:20:51 -- show chassis ambient-temperature
  15:20:59 -- show chassis ethernet-switch
  15:21:13 -- show chassis fan
  15:22:08 -- show version detail
  15:22:17 -- show version brief
  15:22:49 -- show system alarms
  15:24:51 -- show system audit
  15:25:18 -- show system configuration
```

```
  15:25:22 -- show system configuration rescue
  15:26:01 -- show system configuration database
  15:26:04 -- show system configuration database usage
  15:26:14 -- show system configuration archival
  15:26:28 -- show system information
  15:26:50 -- show system login
  15:26:58 -- show system users
  15:28:13 -- request system logout user user_1 terminal pts/0
  15:28:17 -- show system users
  15:28:36 -- show system ntp
  15:28:40 -- show system ntp threshold
  15:28:56 -- show system memory
  15:29:09 -- show system processes
  15:29:14 -- show system processes extensive
  15:31:05 -- restart l2-learning
  15:31:37 -- show system reboot
  15:32:04 -- request system reboot at 21:00
  15:32:11 -- show system reboot
  15:34:35 -- show system rollback
  15:34:40 -- show system rollback compare 1
  15:34:42 -- show system rollback compare 1 2
  15:34:59 -- show system services
  15:35:04 -- show system services service-deployment
  15:35:12 -- show system snapshot
  15:35:18 -- show system software
  15:35:27 -- show system storage
  15:37:13 -- show system memory
  15:38:20 -- show system virtual-memory
  15:38:25 -- show system yang
  15:38:28 -- show system yang package
  15:39:09 -- show cli
  15:39:30 -- show cli authorization
  15:39:57 -- show cli directory
  15:40:02 -- show cli history
```

You have the option to ask for a certain number of commands:

```
user_1@mx960> show cli history 10
  15:35:18 -- show system software
  15:35:27 -- show system storage
  15:37:13 -- show system memory
  15:38:20 -- show system virtual-memory
  15:38:25 -- show system yang
  15:38:28 -- show system yang package
  15:39:09 -- show cli
  15:39:30 -- show cli authorization
  15:39:57 -- show cli directory
  15:40:02 -- show cli history
```

Now what can you change other than the timeout? Here is the list of options:

```
user_1@mx960> set cli ?
Possible completions:
  complete-on-space    Set whether typing space completes current word
  directory            Set working directory
  idle-timeout         Set maximum idle time before login session ends
  logical-system       Set default logical system
  prompt               Set CLI command prompt string
  restart-on-upgrade   Set whether CLI prompts to restart after software upgrade
```

```
  screen-length        Set number of lines on screen
  screen-width         Set number of characters on a line
  terminal             Set terminal type
  timestamp            Timestamp CLI output
```

Some of these might not be that interesting; for instance, why would I want to disable the `complete-on-space`, though there is always a reason why an option is available. There are other options that can be quite handy, however. You can, for example, change the home directory, or the terminal type:

```
ylara@MX960> show cli directory
Current directory: /var/home/pubip

{master:0}[edit system login]
ylara@MX960# run set cli directory /var/home/ylara
Current directory: /var/home/ylara

{master:0}[edit system login]
ylara@MX960> show cli directory
Current directory: /var/home/ylara

user_1@mx960> set cli terminal ?
Possible completions:
  ansi                 ANSI-compatible terminal
  small-xterm          Small (24 line) xterm window
  vt100                VT100-compatible terminal
  xterm                Large (65 line) xterm window
```

You can also change the screen length and width, which might be useful when you are running a script that is requesting a long output and you do not want to deal with the `---more---` prompt, or number of characters per line limit:

```
user_1@mx960> set cli screen-length 0
user_1@mx960> set cli width-length 0
```

The command that you should remember well, and maybe make a habit of using all the time, is `set cli timestamp`, which is going to display a time stamp at the top of any command output. This will help you correlate events and CLI actions for verification, testing, and troubleshooting. In fact, JTAC asks customers to turn this on when collecting information for an open case:

```
ylara@MX960-LR15-VX1.LAX7-re0> show interfaces xe-0/0/0 terse
Interface               Admin Link Proto    Local                 Remote
xe-0/0/0                up    up
xe-0/0/0.0              up    up   aenet    --> ae0.0

ylara@MX960-LR15-VX1.LAX7-re0> set cli timestamp
Apr 30 13:56:10
CLI timestamp set to: %b %d %T

ylara@MX960-LR15-VX1.LAX7-re0> show interfaces xe-0/0/0 terse
Apr 30 13:56:12
Interface               Admin Link Proto    Local                 Remote
xe-0/0/0                up    up
xe-0/0/0.0              up    up   aenet    --> ae0.0
```

Checking the Junos Software Version and Licensing

To check which software version your Junos device is running, issue the `show version` command. This command will also display a long list of software components or sub packages, but in most cases, you will only be interested in checking the first couple of lines:

```
ylara@MX960-LR15-VX1.LAX7-re0> show version
Hostname: MX960-LR15-VX1.LAX7-re0
Model: mx960
Junos: 16.1R7-S6.1
JUNOS OS Kernel 64-bit  [20191025.14243fa_builder_stable_10]
JUNOS OS libs [20191025.14243fa_builder_stable_10]
JUNOS OS runtime [20191025.14243fa_builder_stable_10]
[-- Output omitted --]
```

And if the only information you need at a given time is just those few lines, you can also issue `show system information`:

```
user_1@mx960> show system information
Model: mx960
Family: junos
Junos: 17.4R2-S9.1
Hostname: mx960
```

Not only do you need to have the correct Junos version installed for certain features to function, you also need the proper licensing. To check which licenses are currently installed you can issue the `show system license`:

```
user_1@QFX5100> show system license
License usage:
                                Licenses     Licenses    Licenses    Expiry
  Feature name                       used    installed      needed
  bgp                                   1            1           0    permanent
  isis                                  0            1           0    permanent
  mpls                                  1            0           1    invalid
  vxlan                                 1            1           0    permanent
---more---
```

To install a missing license you would need to obtain the license from your Juniper sales team, and then install it using:

```
user_1@mx960> request system license add terminal
 [Type ^D at a new line to end input,
 enter blank line between each license key]
<your license here>
^D

or
user_1@mx960> request system license add <filename>
```

Router Hardware Components Identification

To check the hardware component of your device, including routing engines, line cards, optics, and so on, you can issue the `show chassis hardware` command.

You can expect the output of this command to be very different when you issue it on different platforms. We will look at some examples.

This first example shows the output on an SRX4200, which is a medium side, fixed-configuration next generation firewall:

```
user@SRX4200> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                SN1234AB0001      SRX4200
Midplane         REV 07   650-071675   12345678901       SRX4200
Routing Engine 0          BUILTIN      BUILTIN           SRX Routing Engine
FPC 0                     BUILTIN      BUILTIN           FEB
  PIC 0                   BUILTIN      BUILTIN           8x10G-SFP
    Xcvr 0       REV 01   740-031980   AA12345678        SFP+-10G-SR
    Xcvr 4       REV 01   740-021308   AB12345678        SFP+-10G-SR
```

The output includes the serial number of the device, lists the SRX Routing Engine as a built-in component, and lists the FPC (Flexible PIC Concentrator) and PIC (Physical Interface Card), which are also built-in. Think of these as built-in line cards with ports.

The SRX4200 comes with eight SFP+ 1G/10G transceiver ports, but the example shows that transceivers are installed only on ports 0 and 4. The output shows the serial number, part number, and the type of transceivers installed. You might want to check this information when you suspect that there are connectivity problems between your device and its neighbor, which might be caused by an incorrect transceiver.

The next example is from an MX960, which is a modular routing platform with 14 slots that can be populated with 11 line cards, and two or three SCBs (Switch Control Boards), which house one or two REs. Because of this modularity and the wide variety of options and combinations, you can expect the output of the `show chassis hardware` command to be long, and to vary significantly from one MX960 installation to another:

```
user_1@mx960-lr4-re1> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                *******           MX960
Midplane         REV 03   710-013698   *******           MX960 Backplane
FPM Board        REV 03   710-014974   *******           Front Panel Display
PDM              Rev 03   740-013110   *******           Power Distribution Module
PEM 0            Rev 05   740-029344   *******           DC 4.1kW Power Entry Module
PEM 1            Rev 05   740-029344   *******           DC 4.1kW Power Entry Module
PEM 2            Rev 05   740-029344   *******           DC 4.1kW Power Entry Module
```

```
PEM 3            Rev 05   740-029344   *******          DC 4.1kW Power Entry Module
Routing Engine 1 REV 01   740-051822   *******          RE-S-1800x4
Routing Engine 1 REV 01   740-051822   *******          RE-S-1800x4
CB 0             REV 10   710-021523   *******          MX SCB
CB 1             REV 10   710-021523   *******          MX SCB
CB 2             REV 10   710-021523   *******          MX SCB
FPC 0            REV 26   750-028467   *******          MPC 3D 16x 10GE
  CPU            REV 10   711-029089   *******          AMPC PMB
  PIC 0                   BUILTIN      *******          4x 10GE(LAN) SFP+
    Xcvr 1       REV 01   740-031980   *******          SFP+-10G-SR
    Xcvr 2       REV 01   740-031980   *******          SFP+-10G-SR
    Xcvr 3       REV 01   740-031980   *******          SFP+-10G-SR
  PIC 1                   BUILTIN      BUILTIN          4x 10GE(LAN) SFP+
    Xcvr 1                NON-JNPR     *******          SFP+-10G-LR
    Xcvr 2       REV 01   740-031980   *******          SFP+-10G-SR
    Xcvr 3       REV 01   740-031980   *******          SFP+-10G-SR
  PIC 2                   BUILTIN      BUILTIN          4x 10GE(LAN) SFP+
    Xcvr 0       REV 01   740-031980   *******          SFP+-10G-SR
    Xcvr 1       REV 01   740-031980   *******          SFP+-10G-SRA
    Xcvr 2       REV 01   740-031980   *******          SFP+-10G-SR
    Xcvr 3       REV 01   740-031980   *******          SFP+-10G-SR
  PIC 3                   BUILTIN      BUILTIN          4x 10GE(LAN) SFP+
FPC 1            REV 17   750-031089   *******          MPC Type 2 3D
  CPU            REV 06   711-030884   *******          MPC PMB 2G
  MIC 0          REV 24   750-028392   *******          3D 20x 1GE(LAN) SFP
    PIC 0                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 0     REV 01   740-011613   *******          SFP-SX
      Xcvr 1     REV 01   740-011782   *******          SFP-SX
    PIC 1                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 0     REV 02   740-011613   *******          SFP-SX
      Xcvr 1     REV 02   740-013111   *******          SFP-T
      Xcvr 3     U        NON-JNPR     *******          SFP-T
      Xcvr 9     REV 01   740-011613   *******          SFP-SX
  MIC 1          REV 24   750-028392   *******          3D 20x 1GE(LAN) SFP
    PIC 2                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 3     REV 01   740-031851   *******          SFP-SX
      Xcvr 4     REV 01   740-031851   *******          SFP-SX
      Xcvr 5     U        NON-JNPR     *******          SFP-SX
      Xcvr 8     REV 02   740-011613   PFP3E48          SFP-SX
    PIC 3                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 1              NON-JNPR     *******          SFP-T
FPC 2            REV 25   750-024064   *******          MS-DPC
  CPU            REV 12   710-013713   *******          DPC PMB
  PIC 0                   BUILTIN      BUILTIN          MS-DPC PIC
  PIC 1                   BUILTIN      BUILTIN          MS-DPC PIC
Fan Tray 0       REV 08   740-031521   *******          Enhanced Fan Tray
Fan Tray 1       REV 08   740-031521   *******          Enhanced Fan Tray
```

You can see in the output that the router has installed three SCBs and two routing engines. It has four PEMs (Power Entry Modules) and two fan trays, as well as line cards in slots 0, 1, and 2.

The line card in slot 0 is an MPC 3D 16x 10GE. The one in slot 1 is a modular MPC Type 2 3D with two 3D 20x 1GE(LAN) SFP MICs. The card on slot 2 is a multiservice DPC used for services such as IPsec, NAT, and GRE tunneling.

There are different types of transceivers installed across the line cards, including SFP+-10G-SR, SFP-SX, and SFP-T. Where SPF stands for *Small-factor Pluggable Transceiver*, SR is *Short Range*, SX stands for *Short Wave Length*, and T is *Twisted-pair* (copper).

We cannot cover the extensive list of line cards and transceiver types supported by all the different Juniper platforms, but this information is available on Juniper Pathfinder: https://apps.juniper.net/home/mx960/hardware-compatibility

When you look at the output, specifically for the FPCs, one question that might come to mind is: how do I map the number I am seeing with the actual port numbers?

For the MPC Type 2 3D in the example, you can see the PFC number, a MIC number, a PIC number, and a transceiver number, which is, yes, confusing:

```
user_1@mx960-lr4-re1> show chassis hardware | find fpc1
FPC 1            REV 17   750-031089   *******          MPC Type 2 3D
  CPU            REV 06   711-030884   *******          MPC PMB 2G
  MIC 0          REV 24   750-028392   *******          3D 20x 1GE(LAN) SFP
    PIC 0                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 0     REV 01   740-011613   *******          SFP-SX
      Xcvr 1     REV 01   740-011782   *******          SFP-SX
    PIC 1                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 0     REV 02   740-011613   *******          SFP-SX
      Xcvr 1     REV 02   740-013111   *******          SFP-T
      Xcvr 3     U        NON-JNPR     *******          SFP-T
      Xcvr 9     REV 01   740-011613   *******          SFP-SX
  MIC 1          REV 24   750-028392   *******          3D 20x 1GE(LAN) SFP
    PIC 2                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 3     REV 01   740-031851   *******          SFP-SX
      Xcvr 4     REV 01   740-031851   *******          SFP-SX
      Xcvr 5     U        NON-JNPR     *******          SFP-SX
      Xcvr 8     REV 02   740-011613   PFP3E48          SFP-SX
    PIC 3                 BUILTIN      BUILTIN          10x 1GE(LAN) SFP
      Xcvr 1               NON-JNPR     *******          SFP-T
```

If you check the `show interface` output on this router, and focus on the ports on slot 1, you will find that the port numbers only include the PFC, the PIC, and the transceiver number. The MIC number is basically ignored:

```
ylara@SA3> show interfaces xe-1* terse | except \. | match up|down
xe-1/0/0                up    up
xe-1/0/1                up    up
xe-1/1/0                up    up
xe-1/1/1                up    up
xe-1/1/3                up    up
xe-1/1/9                up    up
xe-1/2/3                up    up
xe-1/2/4                up    up
xe-1/2/5                up    up
xe-1/2/8                up    up
xe-1/3/1                up    up
```

You can always physically check your router and look at the tiny little labels, or again, check Juniper's web site because, happily, this is also well documented:

- https://www.juniper.net/documentation/en_US/junos/topics/topic-map/chassis-guide-tm-mx-series-mpcs.html#mic-pic-mappingMPC2
- https://www.juniper.net/documentation/en_US/release-independent/junos/topics/reference/general/mic-mx-series-gigabit-ethernet-e-sfp-description.html

Another useful command is `show chassis hardware models`, which shows the actual model number of the hardware component:

```
user_1@mx960-lr4-re1> show chassis hardware models
Hardware inventory:
Item             Version  Part number  Serial number     FRU model number
Midplane         REV 03   710-013698   *******           CHAS-BP-MX960-S
FPM Board        REV 03   710-014974   *******           CRAFT-MX960-S
PEM 0            Rev 05   740-029344   *******           PWR-MX960-4100-DC-S
PEM 1            Rev 05   740-029344   *******           PWR-MX960-4100-DC-S
PEM 2            Rev 05   740-029344   *******           PWR-MX960-4100-DC-S
PEM 3            Rev 05   740-029344   *******           PWR-MX960-4100-DC-S
Routing Engine 1 REV 01   740-051822   *******           RE-S-1800X4-32G-S
CB 0             REV 10   710-021523   *******           SCB-MX960-S
CB 1             REV 10   710-021523   *******           SCB-MX960-S
CB 2             REV 10   710-021523   *******           SCB-MX960-S
FPC 0            REV 26   750-028467   *******           MPC-3D-16XGE-SFPP
FPC 7            REV 17   750-031089   *******           MX-MPC2-3D
  MIC 0          REV 24   750-028392   *******           MIC-3D-20GE-SFP
  MIC 1          REV 24   750-028392   *******           MIC-3D-20GE-SFP
FPC 8            REV 25   750-024064   *******           MS-DPC
Fan Tray 0       REV 08   740-031521   *******           FFANTRAY-MX960-HC-S
Fan Tray 1       REV 08   740-031521   *******           FFANTRAY-MX960-HC-S
```

System Hardware Environment and Status Verification

Now that you know how to check which hardware you have, you'll want to keep an eye on those components and their operational status. You want to make sure the environmental status (power, temperature, CPU, memory) are under acceptable ranges, and you want to make sure your line cards, CBs, and your routing engine are online and your ports are active.

You can achieve this by issuing different options of the `show chassis environment` command. This section looks at some of those options.

The first command to check is the `show chassis craft-interface`. Hardware documentation usually includes a description of the different LED and alarm displays present on your device, which can provide quick information that might guide you in your initial hardware troubleshooting. However, in many cases you do not have

physical access to the router so that you can check these. The `show chassis craft-interface` gives you a text representation of the status of these LEDs, identifies the presence of any alarm, shows which routing engine is the master, and confirms whether your line cards, control boards, power supplies, and fan trays are operational:

```
user_1@mx960> show chassis craft-interface
Front Panel System LEDs:
Routing Engine    0    1
--------------------------
OK                *    *
Fail              .    .
Master            *    .

Front Panel Alarm Indicators:
-----------------------------
Red LED      *
Yellow LED   *
Major relay  *
Minor relay  *

Front Panel FPC LEDs:
FPC    0   1   2   3   4   5   6   7   8   9   10   11
-------------------------------------------------------
Red    .   .   .   .   .   .   .   .   .   .   .   .
Green  .   *   .   .   .   .   .   .   .   .   .   *

CB LEDs:
 CB    0   1   2
----------------
Amber  .   .   .
Green  *   *   *

PS LEDs:
  PS   0   1   2   3
--------------------
Red    *   *   .   .
Green  .   .   *   *

Fan Tray LEDs:
  FT   0   1
------------
Red    .   .
Green  *   *
```

To specifically check your lines cards, control boards, and routing engine there are different options of the `show chassis` command. You can check the state of the line cards installed in the different slots with `show chassis fpc`.

In this next output you want to see that the state is online. For each online card, the output displays the CPU, memory, and temperature. You can enter the command with or without a specific FPC number:

```
user_1@mx960-lr4-re1> show chassis fpc
                     Temp  CPU Utilization (%)   CPU Utilization (%)  Memory
Utilization (%)
Slot State            (C)  Total  Interrupt      1min   5min   15min  DRAM (MB) Heap     Buffer
  0  Online            40     14          0       18     16     15    2048      13         13
  1  Empty
  2  Empty
  3  Empty
  4  Empty
  5  Empty
  6  Empty
  7  Online            36     18          0       16     16     16    2048      11         14
  8  Online            43      3          0        4      4      4    1024      8          41
  9  Empty
 10  Empty
 11  Empty

user_1@mx960-lr4-re1> show chassis fpc 0
                     Temp  CPU Utilization (%)   CPU Utilization (%)  Memory    Utilization (%)
Slot State            (C)  Total  Interrupt      1min   5min   15min  DRAM (MB) Heap     Buffer
  0  Online            40     15          0       15     15     15    2048       13         13
```

If you issue the command with the `detail` option you can see the uptime of your line card and when it was started. It's good to check if the status of your line cards has changed recently:

```
user_1@mx960> show chassis fpc detail
Apr 27 16:10:06
Slot 1 information:
  State                               Online
  Temperature                      47 degrees C / 116 degrees F
  Total CPU DRAM                 3136 MB
  Total RLDRAM                    516 MB
  Total DDR DRAM                 8192 MB
  Start time                          2020-03-04 14:25:11 EST
  Uptime                              54 days, 44 minutes, 55 seconds
  Max power consumption            465 Watts
Slot 3 information:
  State                               Offline
  Reason                              FPC incompatible with SCB
  Temperature                      26 degrees C / 78 degrees F
  Total CPU DRAM                    0 MB
  Total RLDRAM                      0 MB
  Total DDR DRAM                    0 MB
  Max power consumption            265 Watts
Slot 11 information:
  State                               Online
  Temperature                      39 degrees C / 102 degrees F
  Total CPU DRAM                 2048 MB
  Total RLDRAM                    734 MB
  Total DDR DRAM                 3108 MB
  Start time                          2020-03-04 14:18:48 EST
  Uptime                              54 days, 51 minutes, 18 seconds
  Max power consumption            227 Watts
```

For a quick check of the status of your line cards you can issue the `show chassis fpc-status`:

```
{master}
user_1@mx960> show chassis fpc pic-status
Apr 27 16:10:42
Slot 1   Online       MPC7E 3D MRATE-12xQSFPP-XGE-XLGE-CGE
  PIC 0  Online       MRATE-6xQSFPP-XGE-XLGE-CGE
  PIC 1  Online       MRATE-6xQSFPP-XGE-XLGE-CGE
Slot 3   Offline      MS-DPC
Slot 11  Online       MPCE Type 2 3D Q
  PIC 0  Online       2x 10GE  XFP
  PIC 1  Online       2x 10GE  XFP
  PIC 2  Online       2x 10GE  XFP
  PIC 3  Online       2x 10GE  XFP
```

You can also look at the pics within an FPC with `show chassis pic fpc-slot <slot> pic-slot <slot>`. This command is also useful in finding more details about your transceivers that can help you troubleshoot an interface that is not coming up:

```
user_1@mx960-lr4-re1> show chassis pic fpc-slot 7 pic-slot 0
FPC slot 7, PIC slot 0 information:
  Type                            10x 1GE(LAN) SFP
  State                           Online
  PIC version                2.24
  Uptime                        32 days, 5 hours, 14 minutes, 6 seconds

PIC port information:
                        Fiber                   Xcvr vendor       Wave-    Xcvr
  Port Cable type       type  Xcvr vendor       part number       length   Firmware
  0    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J1  850 nm   0.0
  1    GIGE 1000SX      MM    FINISAR CORP.     FTRJ-8519-7D-J2   850 nm   0.0
  2    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J1  850 nm   0.0
  3    GIGE 1000SX      MM    FINISAR CORP.     FTRJ8519P1BNL-J2  850 nm   0.0
  4    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J1  850 nm   0.0
  5    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J1  850 nm   0.0
  6    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J1  850 nm   0.0
  8    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J4  850 nm   0.0
  9    GIGE 1000SX      MM    FINISAR CORP.     FTLF8519P2BNL-J4  850 nm   0.0
```

When you have a modular device with a dual routing engine and you want to check the health of both routing engines, and also figure out which routing engine is acting as master—meaning it is active—you can use `show chassis routing-engine`. This command provides the state of each RE (master/backup) as well as temperature and CPU and memory utilization. You can also check the starting time, uptime, and identify when and why it was last rebooted:

user_1@mx960> **show chassis routing-engine**

```
Routing Engine status:
  Slot 0:
    Current state                  Master
    Election priority              Master (default)
    Temperature                 34 degrees C / 93 degrees F
    CPU temperature             46 degrees C / 114 degrees F
    DRAM                     49108 MB (49152 MB installed)
    Memory utilization           6 percent
```

```
    5 sec CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     1 percent
      Interrupt                  0 percent
      Idle                      98 percent
    1 min CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     2 percent
      Interrupt                  0 percent
      Idle                      98 percent
    5 min CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     2 percent
      Interrupt                  0 percent
      Idle                      98 percent
    15 min CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     2 percent
      Interrupt                  0 percent
      Idle                      98 percent
    Model                        RE-S-2X00x6
    Serial ID                    CAGK2908
    Start time                   2020-03-04 11:29:22 EST
    Uptime                       54 days, 2 hours, 42 minutes, 5 seconds
    Last reboot reason           0x2000:hypervisor reboot
    Load averages:               1 minute   5 minute  15 minute
                                     0.20       0.23       0.26
Routing Engine status:
  Slot 1:
    Current state                Backup
    Election priority            Backup (default)
    Temperature                 34 degrees C / 93 degrees F
    CPU temperature             46 degrees C / 114 degrees F
    DRAM                     49108 MB (49152 MB installed)
    Memory utilization           6 percent
    5 sec CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     0 percent
      Interrupt                  0 percent
      Idle                     100 percent
    Model                        RE-S-2X00x6
    Serial ID                    CAGK8071
    Start time                   2020-03-04 12:21:57 EST
    Uptime                       54 days, 1 hour, 49 minutes, 30 seconds
    Last reboot reason           0x2000:hypervisor reboot
    Load averages:               1 minute   5 minute  15 minute
                                     0.34       0.28       0.20
```

The SCBs have some vital functions within your router: switch data between the line cards, control the chassis, and house the routing engine.

You can check the presence of the SCBs with `show chassis hardware | match SCB`, which will also display the serial number and the SCB type (SCB, SCBE, SCB2, or SCB3):

```
user@mx960> show chassis hardware | match SCB
Item       Version  Part Number  Serial Number  Description
CB0        REV 07 710-021523 ABBC8281   MX SCB
CB1        REV 07 710-021523 ABBC8323   MX SCB
CB2        REV 07 710-021523 ABBD1410   MX SCB

user@mx960> show chassis hardware models | match SCBE
Item       Version  Part Number  Serial Number  Description
CB0        REV 02 750-031391 YE8505   Enhanced MX SCB
CB1        REV 07 710-031391 YL6769   Enhanced MX SCB
CB2        REV 07 710-031391 YE8492   Enhanced MX SCB
```

But if you want to know their status and the uptime, you need to look at `show chassis fabric summary`:

```
user_1@mx960> show chassis fabric summary
Plane   State    Uptime
 0      Online   6 days, 15 hours, 13 minutes, 23 seconds
 1      Online   6 days, 15 hours, 13 minutes, 22 seconds
 2      Online   6 days, 15 hours, 13 minutes, 22 seconds
 3      Online   6 days, 15 hours, 13 minutes, 22 seconds
 4      Spare    6 days, 15 hours, 13 minutes, 22 seconds
 5      Spare    6 days, 15 hours, 13 minutes, 22 seconds
```

Each SCB provides two fabric planes, and depending on the level of redundancy, you will see that some planes will be online and some in spare state. The example shows that the MX960 is using a 2+1 redundancy model.

MORE? If you want more information about the control board you can check the documentation: https://www.juniper.net/documentation/en_US/release-independent/junos/topics/concept/SCB-description.html

For an overall view of the environmental information about your system, you can use `show chassis environment`. Below is an example taken from an MX80 series device and you can see the power entry module (PEM), also known as the power supplies, temperatures, and fan speeds. All this information is important to the operational health of your Juniper hardware:

```
user@MX80> show chassis environment
Class Item                           Status     Measurement
Temp PEM 0                           OK
     PEM 1                           OK
     RE 0 Intake                     OK         34 degrees C / 93 degrees F
     RE 0 Front Exhaust              OK         37 degrees C / 98 degrees F
     RE 0 Rear Exhaust               OK         33 degrees C / 91 degrees F
     Routing Engine                  OK         41 degrees C / 105 degrees F
     Routing Engine CPU              OK         50 degrees C / 122 degrees F
     TFEB 0 QX 0 TSen                OK         37 degrees C / 98 degrees F
     TFEB 0 QX 0 Chip                OK         46 degrees C / 114 degrees F
     TFEB 0 LU 0 TSen                OK         37 degrees C / 98 degrees F
     TFEB 0 LU 0 Chip                OK         51 degrees C / 123 degrees F
```

```
     TFEB 0 MQ 0 TSen              OK         37 degrees C / 98 degrees F
     TFEB 0 MQ 0 Chip              OK         43 degrees C / 109 degrees F
     TFEB 0 TBB PFE TSen           OK         34 degrees C / 93 degrees F
     TFEB 0 TBB PFE Chip           OK         42 degrees C / 107 degrees F
Fans Fan 1                         OK         Spinning at intermediate-speed
     Fan 2                         OK         Spinning at intermediate-speed
     Fan 3                         OK         Spinning at intermediate-speed
     Fan 4                         OK         Spinning at intermediate-speed
     Fan 5                         OK         Spinning at intermediate-speed
```

If you need more information about power capacity and usage, and the conditions of your fan trays, you can enter `show chassis power` and `show chassis fan` respectively:

```
user_1@mx960> show chassis power
PEM 0:
  State:     Present
  DC input:  Out of range (2 feed expected, 0 feed connected)
  DC input:  48.0 V input (0 mV)
  Capacity:  0 W (maximum 4100 W)

PEM 1:
  State:     Present
  DC input:  Out of range (2 feed expected, 0 feed connected)
  DC input:  48.0 V input (0 mV)
  Capacity:  0 W (maximum 4100 W)

PEM 2:
  State:     Online
  DC input:  OK (2 feed expected, 2 feed connected)
  DC input:  48.0 V input (57500 mV)
  Capacity:  4100 W (maximum 4100 W)
  DC output: 684 W (zone 0, 12 A at 57 V, 16% of capacity)

PEM 3:
  State:     Online
  DC input:  OK (2 feed expected, 2 feed connected)
  DC input:  48.0 V input (57000 mV)
  Capacity:  4100 W (maximum 4100 W)
  DC output: 684 W (zone 1, 12 A at 57 V, 16% of capacity)

System:
  Zone 0:
      Capacity:          4100 W (maximum 4100 W)
      Allocated power:   1007 W (3093 W remaining)
      Actual usage:      684 W
  Zone 1:
      Capacity:          4100 W (maximum 4100 W)
      Allocated power:   1060 W (3040 W remaining)
      Actual usage:      684 W
  Total system capacity: 8200 W (maximum 8200 W)
  Total remaining power: 6133 W

user_1@mx960> show chassis fan
      Item                      Status   RPM     Measurement
      Top Tray Fan 1            OK       3769    Spinning at normal speed
      Top Tray Fan 2            OK       3769    Spinning at normal speed
      Top Tray Fan 3            OK       3790    Spinning at normal speed
      Top Tray Fan 4            OK       3790    Spinning at normal speed
      Top Tray Fan 5            OK       3769    Spinning at normal speed
```

```
      Top Tray Fan 6            OK       3769    Spinning at normal speed
      Top Tray Fan 7            OK       3769    Spinning at normal speed
      Top Tray Fan 8            OK       3790    Spinning at normal speed
      Top Tray Fan 9            OK       3769    Spinning at normal speed
      Top Tray Fan 10           OK       3790    Spinning at normal speed
      Top Tray Fan 11           OK       3790    Spinning at normal speed
      Top Tray Fan 12           OK       3769    Spinning at normal speed
      Bottom Tray Fan 1         OK       3769    Spinning at normal speed
      Bottom Tray Fan 2         OK       3832    Spinning at normal speed
      Bottom Tray Fan 3         OK       3790    Spinning at normal speed
      Bottom Tray Fan 4         OK       3811    Spinning at normal speed
      Bottom Tray Fan 5         OK       3705    Spinning at normal speed
      Bottom Tray Fan 6         OK       3790    Spinning at normal speed
      Bottom Tray Fan 7         OK       3769    Spinning at normal speed
      Bottom Tray Fan 8         OK       3769    Spinning at normal speed
      Bottom Tray Fan 9         OK       3769    Spinning at normal speed
      Bottom Tray Fan 10        OK       3769    Spinning at normal speed
      Bottom Tray Fan 11        OK       3769    Spinning at normal speed
      Bottom Tray Fan 12        OK       3790    Spinning at normal speed
```

Checking for System and Chassis Alarms

When you are suspicious about hardware issues, one good starting point is checking for the presence of chassis-level alarms, which will be displayed when you issue `show chassis alarms`. Any existing alarm will be displayed with the time, the alarm, when the trigger condition was detected and the alarm was raised, the class (minor or major), and a brief description. In this example, you can see that the router currently has a few alarms related to the power supplies that were issued on 2020-04-20:

```
user_1@mx960> show chassis alarms
6 alarms currently active
Alarm time               Class  Description
2020-04-20 23:59:26 EDT  Minor  PEM 1 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 1 Input Failure
2020-04-20 23:59:26 EDT  Major  PEM 1 Not OK
2020-04-20 23:59:26 EDT  Minor  PEM 0 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 0 Input Failure
```

When you find that an alarm is present, you can enter more specific commands to investigate. In this case you would issue the `show chassis power` command:

```
user_1@mx960> show chassis power
PEM 0:
  State:     Present
  DC input:  Out of range (2 feed expected, 0 feed connected)
  DC input:  48.0 V input (0 mV)
  Capacity:  0 W (maximum 4100 W)

PEM 1:
  State:     Present
  DC input:  Out of range (2 feed expected, 0 feed connected)
  DC input:  48.0 V input (0 mV)
  Capacity:  0 W (maximum 4100 W)

PEM 2:
```

```
  State:     Online
  DC input:  OK (2 feed expected, 2 feed connected)
  DC input:  48.0 V input (57500 mV)
  Capacity:  4100 W (maximum 4100 W)
  DC output: 684 W (zone 0, 12 A at 57 V, 16% of capacity)

PEM 3:
  State:     Online
  DC input:  OK (2 feed expected, 2 feed connected)
  DC input:  48.0 V input (57000 mV)
  Capacity:  4100 W (maximum 4100 W)
  DC output: 627 W (zone 1, 11 A at 57 V, 15% of capacity)

System:
  Zone 0:
      Capacity:          4100 W (maximum 4100 W)
      Allocated power:   1007 W (3093 W remaining)
      Actual usage:      684 W
  Zone 1:
      Capacity:          4100 W (maximum 4100 W)
      Allocated power:   1060 W (3040 W remaining)
      Actual usage:      627 W
  Total system capacity: 8200 W (maximum 8200 W)
  Total remaining power: 6133 W
```

You can also issue the `show system alarm`, which will display both chassis-level and system-level alarms. While there is a long list of possible chassis-level alarms, which you can find listed in the documentation in the links below, system alarms are limited to conditions such as missing licenses, or the absence of the rescue configuration:

- https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/show-chassis-alarms.html
- https://www.juniper.net/documentation/en_US/junos/topics/topic-map/chassis-guide-tm-error-handling-alarms.html

The next example shows that three alarms are present: the first two are from the same chassis-level power related alarms that we saw before, and that are still present in our device, and the third one is a system-level alarm indicating that the rescue configuration has not been saved:

```
user_1@mx960> show system alarms
2 alarms currently active
Alarm time               Class  Description
2020-04-20 23:59:26 EDT  Minor  PEM 0 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 0 Input Failure
2020-04-20 23:35:51 UTC  Minor  Rescue configuration is not set
```

Alarms cannot be cleared. The condition that triggered the alarm must be corrected for the alarm to go away.

In the case of the rescue configuration missing alarm, creating this file would clear the condition.

```
user_1@mx960> show system configuration rescue
error: No rescue configuration is set.

user_1@mx960> request system configuration rescue  save
user_1@mx960>

user_1@mx960> show system alarms | match rescue
user_1@mx960>
```

You can also configure your router to generate alarms when an interface fails:

```
user_1@mx960# set chassis alarm ?
Possible completions:
+ apply-groups         Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
> ds1                  DS1 alarms
> ethernet             Ethernet alarms
> integrated-services  Integrated services alarms
> management-ethernet  Management Ethernet alarms
> otn-odu              OTN ODU alarms
> otn-otu              OTN OTU alarms
> serial               Serial alarms
> services             Services PIC alarms
> sonet                SONET alarms
> t3                   DS3 alarms
```

When any of these alarms are present, it will be displayed by either `show chassis alarms` or `show system alarms`:

```
user_1@mx960# set chassis alarm ethernet link-down yellow

user_1@mx960> show chassis alarms
20 alarms currently active
Alarm time               Class  Description
2020-04-30 20:27:35 EDT  Minor  et-1/0/5: Link down
2020-04-30 20:27:35 EDT  Minor  xe-11/3/1: Link down
2020-04-20 23:59:26 EDT  Minor  PEM 1 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 1 Input Failure
2020-04-20 23:59:26 EDT  Major  PEM 1 Not OK
2020-04-20 23:59:26 EDT  Minor  PEM 0 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 0 Input Failure
2020-04-20 23:59:26 EDT  Major  PEM 0 Not OK

@mx960-lr17-re0> show system alarms
20 alarms currently active
Alarm time               Class  Description
2020-04-30 20:27:35 EDT  Minor  et-1/0/5: Link down
2020-04-30 20:27:35 EDT  Minor  xe-11/3/1: Link down
2020-04-20 23:59:26 EDT  Minor  PEM 1 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 1 Input Failure
2020-04-20 23:59:26 EDT  Major  PEM 1 Not OK
2020-04-20 23:59:26 EDT  Minor  PEM 0 Fan Failed
2020-04-20 23:59:26 EDT  Major  PEM 0 Input Failure
2020-04-20 23:59:26 EDT  Major  PEM 0 Not OK
```

Alarms will trigger a corresponding alarm LED on the front panel of your device, with major alarms showing red and minor alarms showing orange. Clearing these alarms (by resolving the presented issue) will also clear the alarm LED.

Checking for System Core Dumps

When a Junos process has crashed or terminated abnormally it will record the state of memory and some additional information that would help with diagnosing the root cause of the crash. The standard procedure is to open a JTAC case and have them analyze the core files. You want to keep track of any other symptoms observed at the time of the crash, as well as which action was taken (like configuration changes, software upgrade, additional hardware) that might have triggered it.

To check if your device has generated a core dump file use the `show system core-dumps` command:

```
user@vmx> show system core-dumps
-rw-r--r--  1 root  wheel    1849777 Mar 10 07:09 /var/crash/core.vmxt.mpc0.1583824131.3262.gz
/var/tmp/*core*: No such file or directory
/var/tmp/pics/*core*: No such file or directory
/var/crash/kernel.*: No such file or directory
/var/jails/rest-api/tmp/*core*: No such file or directory
/tftpboot/corefiles/*core*: No such file or directory

total files: 3
```

Verifying System Processes

To display information about the software processes that are running on the router or switch you can issue the `show system processes` command. This output is very long and can be a little overwhelming. And honestly, unless you are dealing with some very specific issue for which it's likely you have a case opened, you won't use this command that often. However, you should be familiar with some very common processes: rpd, chassid, snmpd, eventd l2ald, mdg:

```
user_1@mx960> show system processes extensive
last pid: 90857;  load averages:  0.51,  0.35,  0.28  up 54+03:00:36    15:29:58
301 processes: 5 running, 253 sleeping, 43 waiting
Mem: 73M Active, 5342M Inact, 2153M Wired, 1571M Buf, 39G Free
Swap: 3072M Total, 3072M Free
  PID USERNAME    PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root        155 ki31     0K    64K CPU1    1 1281.1 100.00% idle{idle: cpu1}
   11 root        155 ki31     0K    64K RUN     2 1280.8 100.00% idle{idle: cpu2}
   11 root        155 ki31     0K    64K CPU3    3 1280.7 100.00% idle{idle: cpu3}
   11 root        155 ki31     0K    64K CPU0    0 1279.1 100.00% idle{idle: cpu0}
73061 root         23    0   818M 59932K select  2 563:41   5.27% chassisd{chassisd}
73030 root         20    0  1145M   200M kqread  3 145:44   0.00% rpd{TraceThread}
   12 root        -60    -     0K   688K WAIT    0  81:41   0.00% intr{swi4: clock (0)}
73030 root         20    0  1145M   200M kqread  2  75:30   0.00% rpd{rpd}
   12 root        -72    -     0K   688K WAIT    1  72:14   0.00% intr{swi1: netisr 0}
72969 root         20    0   721M 11624K select  3  67:17   0.00% clksyncd
---more---
```

Also be aware of the fact that the modularity of Junos allows restarting a particular process without affecting other processes. Though rare, you might need to restart a particular process to clear a condition:

```
user_1@mx960> show system processes extensive | match dcd
 1941 root               1  96    0 72396K 13100K select 124:56  0.00% dcd

user_1@mx960> restart interface-control
Interface control process started, pid 4684

user_1@mx960> show system processes extensive | match dcd
 4684 root               1 108    0 72464K 14220K select   0:01 31.51% dcd
```

Checking System Current Time and Uptime

Another common task in network operations is to check the system uptime, which can help detect any hardware failure that might have caused, for example, a RE failure. This is easily validated issuing the `show system uptime` command:

```
user@vSRX1-JNCIA> show system uptime
Current time: 2020-04-21 20:41:07 UTC
Time Source:  NTP CLOCK
System booted: 2020-04-20 23:33:34 UTC (21:07:33 ago)
Protocols started: 2020-04-20 23:35:51 UTC (21:05:16 ago)
Last configured: 2020-04-21 13:11:28 UTC (07:29:39 ago) by jncia
 8:41PM  up 21:08, 1 users, load averages: 1.23, 1.33, 1.32
```

You can check the current time and times one on the system, and also when it was last rebooted, when was the last time protocols were started, and when and who made changes in the box the last time. All useful information when troubleshooting sudden behavior changes. The output also shows that the time source is NTP.

It is important and a good practice to make sure all systems are synchronized so that events, logs, and alarms can be correlated. You can set the time on your Junos device manually with the `set date` command, but the best practice is to use NTP:

```
user_1@mx960> set date ?
Apr 27 15:49:26
Possible completions:
  <time>               New date and time (YYYYMMDDhhmm.ss)
  ntp                  Set system date and time using Network Time Protocol servers
```

You can check the status of NTP and whether your device is synchronized with an NTP server, with `show ntp status` and `show ntp associations`:

```
user_1@mx960> show ntp status
Apr 27 15:43:49
status=0644 leap_none, sync_ntp, 4 events, event_peer/strat_chg,
version=ntpd 4.2.0-a Thu Jan 30 13:14:42  2020 (1), processor=amd64,
system=FreeBSDJNPR-11.0-20191203.fa5e90e_buil, leap=00, stratum=4,
precision=-23, rootdelay=14.612, rootdispersion=35.133, peer=64516,
refid=153.39.123.254,
```

```
reftime=e251b440.8185682f  Mon, Apr 27 2020 15:42:56.505, poll=6,
clock=e251b475.baf840a1  Mon, Apr 27 2020 15:43:49.730, state=4,
offset=0.454, frequency=56.991, jitter=1.086, stability=0.009

{master}
user_1@mx960> show ntp associations
Apr 27 15:43:54
   remote         refid           st t when poll reach   delay   offset  jitter
==============================================================================
*GWR-V4.vzbi.com  166.37.130.87    3 -   58   64  377    0.711    0.454   1.450
```

Connected Users and Change Management

There will be occasions when you will want to know who is connected to the device, and whether they are active at that time. You can easily check that with `show system users`:

```
user_3@mx960> show system users
 3:26PM  up 54 days,  2:58, 2 users, load averages: 0.27, 0.32, 0.27
USER     TTY      FROM                              LOGIN@  IDLE WHAT
user_1    pts/0    10.67.22.15                     Wed12PM 33:20 -cl
user_3    pts/1    10.67.22.197                    3:01PM      - -cl
```

In this example, you log in as user_3 and when you issue the `show system users` you find that user_1 has been idle for more than a day. When you enter configuration mode you also get a message indicating that this user is still also in configuration mode:

```
{master}
user_3@mx960> edit
Entering configuration mode
Users currently editing the configuration:
  User_1 terminal pts/1 (pid 30886) on since 2020-04-29 12:50:04 EDT, idle 33:13:01
      {master}[edit]
```

If you need to close this user's connection you can enter:

```
{master}
user_1@mx960> request system logout user user_1 terminal pts/0
logout-user: done

{master}
user_1@mx960> show system users
3:26PM  up 54 days,  2:58, 2 users, load averages: 0.27, 0.32, 0.27
USER     TTY      FROM                              LOGIN@  IDLE WHAT
user_3    pts/1    10.67.22.197                    3:01PM      - -cl
```

You might also find that changes have been made and you want to know who made the changes, and when. To check the commit history of your device you can use the `show system commit` command:

```
user_3@mx960> show system commit
0   2020-04-29 09:11:28 UTC by user_1 via cli
1   2020-04-29 08:09:07 UTC by user_1 via cli
2   2020-04-20 23:56:46 UTC by root via cli
3   2020-04-20 23:31:32 UTC by root via cli
```

You can see in the output that user_1, the one that you just disconnected, made changes yesterday. You can issue the command `show | compare` to see what changes were made:

```
[edit]
user_3@mx960# show | compare rollback 2
 [edit interfaces]
-   unit 1 {
-       vlan-id 1;
-       family inet {
-           address 5.5.5.9/30;
-       }
-   }
+   unit 2 {
+       vlan-id 2;
+       family inet {
+           address 5.5.5.10/30;
+       }
+       family iso;
+       family mpls;
+   }
```

You now know that the technician changed the configuration of interface ge-0/0/0 by removing unit 1 and adding unit 2 with a new VLAN and a new IP address. Depending on the situation you might want to remove the changes by issuing a `rollback 2` command.

Interface Monitoring

Checking interface status is probably one of the most common tasks for troubleshooting, verification, maintenance, or monitoring. Usually you want to know two things: the status of the interface and its IP address. You might also want to check other attributes, check traffic statistics, and look at the packets coming in and leaving your interface.

Interface Status, IP Addresses, and Other Attributes

The quickest and easiest way to obtain just the addresses and status of an interface is to append `terse` to the end of a command. The `terse` option significantly reduces the amount of information returned and can be used with several commands. In this case, we will use it to get a condensed version of the `show interfaces` command.

In the next example, we are using the terse options and also filtering the output using `ge*`, which only displays ge interfaces:

```
user_1@mx960> show interfaces terse ge*
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.100            up    up   inet     192.168.1.1/24
                                  iso
                                  inet6    2001:db8:192:168:1::1/64
                                           fe80::f21c:2dff:feda:a100/64
ge-0/0/0.200            up    up   inet     192.168.10.1/24

ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     192.168.2.1/24
                                           192.168.2.2/24
                                           192.168.12.1/24
                                  iso
                                  mpls

ge-0/0/2                down  down
ge-0/0/2.0              up    down inet     192.168.3.1/24

ge-0/0/3                up    up
ge-0/0/3.0              down  up   inet     192.168.4.1/24

ge-0/0/4                up    up
ge-0/0/5                up    up
ge-0/0/6                up    up
ge-0/0/7                up    up
```

Although this output looks short and simple, there is plenty of information for us to digest.

The first thing you can see is that the first four interfaces are listed as ge-x.y.z and also as ge-x.y.z.number. We covered this in detail in Chapter 2, but you can analyze the details shown in the example for each interface, which will serve as a review: the first entry represents the physical interface, and second entry represents a logical interface or unit number within the physical interface.

For example, ge-0/0/0.100 is a logical interface within ge-0/0/0. The unit number can only be a number different than 0, when the interface is doing VLAN-tagging, and in that case, the interface can only have more than one unit. Here is how ge-0/0/0 is actually configured:

```
[edit interfaces ge-0/0/0]
user_1@mx960# show | display set
set interfaces ge-0/0/0 vlan-tagging
set interfaces ge-0/0/0 unit 100 vlan-id 100
set interfaces ge-0/0/0 unit 100 family inet address 192.168.1.1/24
set interfaces ge-0/0/0 unit 100 family iso
set interfaces ge-0/0/0 unit 100 family inet6 address 2001:db8:192:168:1::1/64
set interfaces ge-0/0/0 unit 100 family mpls
set interfaces ge-0/0/0 unit 200 vlan-id 200
set interfaces ge-0/0/0 unit 200 family inet address 192.168.10.1/24
```

All the logical properties, like the IP addresses of an interface, are configured within the logical interface.

You can also see from the output for ge-0/0/0.100 that there could be more than one protocol configured under the same interface. In this case, we have IPv4 (inet), IPv6 (inet6), MPLS, and ISO (used for ISIS).

Notice that the interface has two IPv6 addresses. 2001:db8:192:168:1::1/64 is a global IPv6 address that was configured by us, and fe80::f21c:2dff:feda:a100/64 is the interfaces link-local address that was automatically created using the interfaces MAC address, which you can find by typing:

```
user_1@mx960> show interfaces ge-0/0/0 | match hardware
  Current address: f0:1c:2d:da:a1:00, Hardware address: f0:1c:2d:da:a1:00
```

When you look at ge-0/0/1.0, you realize that this interface has three IPv4 addresses. Remember that this is allowed by Junos and that one address per subnet will be selected as preferred, and one address out of all will be selected as primary:

```
user_1@mx960> show interfaces ge-0/0/1.0 |find addresses
      Addresses, Flags: Is-Default Is-Preferred Is-Primary
        Destination: 192.168.2/24, Local: 192.168.2.1, Broadcast: 192.168.2.255
      Addresses
        Destination: 192.168.2/24, Local: 192.168.2.2, Broadcast: 192.168.2.255
      Addresses, Flags: Is-Preferred
        Destination: 192.168.12/24, Local: 192.168.12.1,
        Broadcast: 192.168.12.255
    Protocol iso, MTU: 1497
      Flags: Is-Primary
    Protocol mpls, MTU: 1488, Maximum labels: 3
      Flags: Is-Primary
```

Interfaces ge-0/0/1 and ge-0/0/2 only have single IPv4 addresses, but they have some interesting interface status combinations. When you look at the `shows interfaces terse` command you can see a column showing the administrative status, and a column showing the actual interface status (whether it is physically up or not).

All interfaces are administratively `up` by default, and they come up automatically as long as there is a link, in other words, as long as there is an active device connected to that interface. Thus, when you look at these two interfaces and see down under admin, you know that someone manually shutdown or disabled the interface. This can be done in two ways:

```
# set interface <interface> [<unit#>] disable
```

or

```
% ifconfig <interface> down (at the shell prompt)
```

When you look at the interfaces again, you notice that ge-0/0/2 shows only the root user gets into the UNIX shell admin status `down`, link status `down`, while the logical interface ge-0/0/2.0 shows admin status `up`, link `down`:

```
ge-0/0/2             down  down
ge-0/0/2.0            up   down inet    192.168.3.1/24
```

This could be strange at first, but look at it this way:

If we do: `set interface ge-0/0/2 disable`. We are shutting down interface ge-0/0/2, and we are doing that manually (administratively). Shutting it down means literally turning it off! Thus the interface goes physically down (see Figure 4.1). Any logical interface within the interface goes down as well.

Figure 4.1 Example of Disable

However, since we never did `set interface ge-0/0/2 unit 0 disable`, the logical interface ge-0/0/0.2 is NOT administratively down.

The second case is a little bit trickier.

The nge-0/0/3 shows admin status `up`, link status `up`, while the logical interface ge-0/0/3.0 shows admin status `down`, link `up`. Why is that?

```
ge-0/0/3                up    up
ge-0/0/3.0              down  up    inet     192.168.4.1/24
```

If we do: `set interface ge-0/0/3 unit 0 disable` , we are disabling the logical interface, but since the physical interface is still up (able to transmit/receive bits), the logical interface remains physically up (See Figure 4.2). What you are disabling is the logical properties assigned to that interface.

Figure 4.2 Example of Logical Interface

You can think of it this way: before you disable the logical interface, there are local and direct routes associated with that interface:

```
user_1@mx960> show route terse table inet.0 | match 192.168.4
* 192.168.4.0/24     D   0                    >ge-0/0/3.0
* 192.168.4.1/32     L   0                     Local

[edit interfaces ge-0/0/3 unit 0]
user_1@mx960# set disable

[edit interfaces ge-0/0/3 unit 0]
user_1@mx960# commit
commit complete
```

After you disable the logical interface the routes are gone, even though the interface is still up:

```
user_1@mx960> show route protocol direct terse | match 192.168.4
```

Now you know how to check the status and interface addresses, but what if you need more information? You might need to know the MAC address, the MTU, the SNMP index, or whether there is any traffic going through the interface. In this case, you will need to enter `show interface` or, even, `show interface extensive`:

The output of these commands will be long! So, you will need to put in practice all the different ways to filter the output. First, you can just request information for a specific interface:

```
user_1@mx960> show interfaces ge-0/0/0
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 134, SNMP ifIndex: 508
  Link-level type: Ethernet, MTU: 1518, Link-mode: Full-duplex, Speed: 100mbps,
  BPDU Error: None, MAC-REWRITE Error: None, Loopback: Disabled,
  Source filtering: Disabled, Flow control: Enabled
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x0
  CoS queues     : 8 supported, 8 maximum usable queues
  Current address: f0:1c:2d:da:a1:00, Hardware address: f0:1c:2d:da:a1:00
  Last flapped   : 2020-03-26 21:58:11 UTC (4w6d 01:31 ago)
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
  Active alarms  : None
  Active defects : None
  Interface transmit statistics: Disabled

  Logical interface ge-0/0/0.100 (Index 77) (SNMP ifIndex 543)
    Flags: SNMP-Traps 0x0 VLAN-Tag [ 0x8100.100 ]  Encapsulation: ENET2
    Input packets : 0
    Output packets: 9
    Protocol inet, MTU: 1500
      Flags: Sendbcast-pkt-to-re
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.1/24, Local: 192.168.1.1, Broadcast: 192.168.1.255
    Protocol iso, MTU: 1497
      Flags: None
    Protocol inet6, MTU: 1500
      Flags: Is-Primary
      Addresses, Flags: Is-Default Is-Preferred Is-Primary
        Destination: 2001:db8:192:168::/64, Local: 2001:db8:192:168:1::1
      Addresses, Flags: Is-Preferred
        Destination: fe80::/64, Local: fe80::f21c:2d00:64da:a100
    Protocol mpls, MTU: 1488, Maximum labels: 3

  Logical interface ge-0/0/0.200 (Index 72) (SNMP ifIndex 545)
    Flags: SNMP-Traps 0x0 VLAN-Tag [ 0x8100.200 ]  Encapsulation: ENET2
    Input packets : 0
    Output packets: 1
    Protocol inet, MTU: 1500
      Flags: Sendbcast-pkt-to-re
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 192.168.10/24, Local: 192.168.10.10,
        Broadcast: 192.168.10.255
```

```
  Logical interface ge-0/0/0.32767 (Index 78) (SNMP ifIndex 544)
    Flags: SNMP-Traps 0x0 VLAN-Tag [ 0x0000.0 ]  Encapsulation: ENET2
    Input packets : 0
    Output packets: 0
    Security: Zone: Null
```

Let's say you need to find the MAC address, SNMP index, and the MTU. You can use the `| match` option to filter the output:

```
user_1@mx960> show interfaces ge-0/0/0 | match hardware|mtu|index
  Interface index: 134, SNMP ifIndex: 508
  Link-level type: Ethernet, MTU: 1518, Link-mode: Full-duplex, Speed: 100mbps,
  Current address: f0:1c:2d:da:a1:00, Hardware address: f0:1c:2d:da:a1:00
  Logical interface ge-0/0/0.100 (Index 77) (SNMP ifIndex 543)
    Protocol inet, MTU: 1500
    Protocol iso, MTU: 1497
    Protocol inet6, MTU: 1500
    Protocol mpls, MTU: 1488, Maximum labels: 3
  Logical interface ge-0/0/0.200 (Index 72) (SNMP ifIndex 545)
    Protocol inet, MTU: 1500
  Logical interface ge-0/0/0.32767 (Index 78) (SNMP ifIndex 544)
```

Remember that `| match, find, except` and so on, can help you find the specific information that you need much quicker than having to look through dozens of output lines. This is true for both operational mode and configuration mode. Practice using it every chance you get. Here are some additional examples:

```
user_1@mx960> show interfaces ge-0/* terse| match up.*up
ge-0/0/0                up    up
ge-0/0/0.100            up    up   inet     192.168.1.1/24
ge-0/0/0.200            up    up   inet     192.168.10.10/24
ge-0/0/0.32767          up    up
ge-0/0/1                up    up
ge-0/0/2                up    up
ge-0/0/2.0              up    up   inet     192.168.3.1/24
ge-0/0/3                up    up
ge-0/0/3.0              up    up   inet     192.168.4.1/24
ge-0/0/4                up    up
ge-0/0/5                up    up
ge-0/0/6                up    up
ge-0/0/7                up    up

user_1@mx960> show interfaces ge-0/* terse| match up.*up | except \.
ge-0/0/0                up    up
ge-0/0/1                up    up
ge-0/0/2                up    up
ge-0/0/3                up    up
ge-0/0/4                up    up
ge-0/0/5                up    up
ge-0/0/6                up    up
ge-0/0/7                up    up

[edit interfaces]
ylara@R1# show| display set | match ge-0/0/0
set interfaces ge-0/0/0 vlan-tagging
set interfaces ge-0/0/0 unit 100 vlan-id 100
set interfaces ge-0/0/0 unit 100 family inet address 192.168.1.1/24
```

```
set interfaces ge-0/0/0 unit 100 family iso
set interfaces ge-0/0/0 unit 100 family inet6 address 2001:db8:192:168:1::1/64
set interfaces ge-0/0/0 unit 100 family mpls
set interfaces ge-0/0/0 unit 200 vlan-id 200
set interfaces ge-0/0/0 unit 200 family inet address 192.168.10.10/24

user_1@mx960> show route 192.168/16 terse table inet.0
inet.0: 13 destinations, 15 routes (13 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 192.168.1.0/24     D   0                      >fe-0/0/0.100
* 192.168.1.1/32     D   0                      >lo0.1
                     L   0                       Local
* 192.168.2.0/24     D   0                      >fe-0/0/1.0
                     D   0                      >fe-0/0/1.0
* 192.168.2.1/32     L   0                       Local
* 192.168.2.2/32     L   0                       Local
* 192.168.3.0/24     D   0                      >fe-0/0/2.0
* 192.168.3.1/32     L   0                       Local
* 192.168.4.0/24     D   0                      >fe-0/0/3.0
* 192.168.4.1/32     L   0                       Local
* 192.168.10.0/24    D   0                      >fe-0/0/0.200
* 192.168.10.10/32   L   0                       Local
* 192.168.12.0/24    D   0                      >fe-0/0/1.0
* 192.168.12.1/32    L   0                       Local

user_1@mx960> show route 192.168/16 terse table inet.0 | match /32
* 192.168.1.1/32     D   0                      >lo0.1
* 192.168.2.1/32     L   0                       Local
* 192.168.2.2/32     L   0                       Local
* 192.168.3.1/32     L   0                       Local
* 192.168.4.1/32     L   0                       Local
* 192.168.10.10/32   L   0                       Local
* 192.168.12.1/32    L   0                       Local
```

Interface Statistics

When you need to check interface statistics there are different ways to do so, though keep in mind each one provides different levels of information.

- `show interfaces <interface> statistics`, or just `show interfaces <interface>`, which gives you a snapshot of some of the interface counters
- `show interfaces <interface> extensive`, which gives you a snapshot of the interface counters
- `monitor interface <interface>`, which provides real-time counters
- `monitor interface traffic`

The first two options will display the interface statistics at the time you issue the command:

```
user_1@mx960> show interfaces ge-0/0/0
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 149, SNMP ifIndex: 527
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, LAN-PHY mode, Speed: 1000mbps, BPDU Error: None,
Loop Detect PDU Error: None,
  Ethernet-Switching Error: None, MAC-REWRITE Error: None, Loopback: Disabled, Source filtering:
Disabled, Flow control: Enabled,
  Auto-negotiation: Enabled, Remote fault: Online
  Pad to minimum frame size: Disabled
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x4000
  CoS queues     : 8 supported, 8 maximum usable queues
  Current address: 00:50:56:a2:d0:80, Hardware address: 00:50:56:a2:d0:80
  Last flapped   : 2020-05-02 01:20:22 UTC (01:09:10 ago)
  Input rate     : 0 bps (0 pps)
  Output rate    : 0 bps (0 pps)
  Active alarms  : None
  Active defects : None
  PCS statistics                      Seconds
    Bit errors                             0
    Errored blocks                         0
   Ethernet FEC statistics              Errors
    FEC Corrected Errors                    0
    FEC Uncorrected Errors                  0
    FEC Corrected Errors Rate               0
    FEC Uncorrected Errors Rate             0
   Interface transmit statistics: Enabled

  Logical interface ge-0/0/0.0 (Index 340) (SNMP ifIndex 543)
    Flags: Up SNMP-Traps 0x4004000 Encapsulation: ENET2
    Input packets : 525
    Output packets: 552
    Protocol inet, MTU: 1500
    Max nh cache: 75000, New hold nh limit: 75000, Curr nh cnt: 1, Curr new hold cnt: 0, NH drop cnt: 0
      Flags: Sendbcast-pkt-to-re
      Addresses, Flags: Is-Preferred Is-Primary
        Destination: 10.100.12/24, Local: 10.100.12.1, Broadcast: 10.100.12.255
    Protocol multiservice, MTU: Unlimited
      Flags: Is-Primary
```

As you can see, these are very basic statistics. Let's take a look at what we get when we use extensive:

```
user_1@mx960# run show interfaces ge-0/0/0 extensive
Physical interface: ge-0/0/0, Enabled, Physical link is Up
  Interface index: 149, SNMP ifIndex: 527, Generation: 152
  Link-level type: Ethernet, MTU: 1514, MRU: 1522, LAN-
PHY mode, Speed: 1000mbps, BPDU Error: None, Loop Detect PDU Error: None,
  Ethernet-Switching Error: None, MAC-
REWRITE Error: None, Loopback: Disabled, Source filtering: Disabled, Flow control: Enabled,
  Auto-negotiation: Enabled, Remote fault: Online
  Pad to minimum frame size: Disabled
  Device flags   : Present Running
  Interface flags: SNMP-Traps Internal: 0x4000
  CoS queues     : 8 supported, 8 maximum usable queues
  Hold-times     : Up 0 ms, Down 0 ms
  Damping        : half-life: 0 sec, max-suppress: 0 sec, reuse: 0, suppress: 0, state: unsuppressed
  Current address: 00:50:56:a2:d0:80, Hardware address: 00:50:56:a2:d0:80
```

```
  Last flapped   : 2020-05-02 01:20:22 UTC (01:11:11 ago)
  Statistics last cleared: Never
  Traffic statistics:
   Input  bytes  :                44456                    0 bps
   Output bytes  :                    0                    0 bps
   Input  packets:                  548                    0 pps
   Output packets:                    0                    0 pps
   IPv6 transit statistics:
   Input  bytes  :                    0
   Output bytes  :                    0
   Input  packets:                    0
   Output packets:                    0
  Dropped traffic statistics due to STP State:
   Input  bytes  :                    0
   Output bytes  :                    0
   Input  packets:                    0
   Output packets:                    0
  Input errors:
   Errors: 0, Drops: 0, Framing errors: 0, Runts: 0, Policed discards: 0, L3 incompletes: 0, L2 channel
errors: 0, L2 mismatch timeouts: 0,
   FIFO errors: 0, Resource errors: 0
 Output errors:
   Carrier transitions: 3, Errors: 0, Drops: 0, Collisions: 0, Aged packets: 0, FIFO errors: 0, HS link
CRC errors: 0, MTU errors: 0,
    Resource errors: 0
  Active alarms  : None
  Active defects : None
  PCS statistics                      Seconds
    Bit errors                             0
    Errored blocks                         0
  Ethernet FEC statistics              Errors
    FEC Corrected Errors                    0
    FEC Uncorrected Errors                  0
    FEC Corrected Errors Rate               0
    FEC Uncorrected Errors Rate             0
  MAC statistics:                      Receive         Transmit
    Total octets                         52166            57056
    Total packets                          549              570
    Unicast packets                        549              570
    Broadcast packets                        0                0
    Multicast packets                        0                0
    CRC/Align errors                         0                0
    FIFO errors                              0                0
    MAC control frames                       0                0
    MAC pause frames                         0                0
    Oversized frames                         0
    Jabber frames                            0
    Fragment frames                          0
    VLAN tagged frames                       0
    Code violations                          0
    Total errors                             0                0
  Filter statistics:
    Input packet count                     549
    Input packet rejects                     8
    Input DA rejects                         0
    Input SA rejects                         0
    Output packet count                                     569
```

```
  Output packet pad count                              0
  Output packet error count                                    0
   CAM destination filters: 0, CAM source filters: 0
Autonegotiation information:
  Negotiation status: Incomplete
Packet Forwarding Engine configuration:
  Destination slot: 0 (0x00)
CoS information:
  Direction : Output
  CoS transmit queue               Bandwidth               Buffer Priority   Limit
                            %            bps     %            usec
  0 best-effort            95      950000000    95               0      low    none
  3 network-control         5       50000000     5               0      low    none
Interface transmit statistics: Enabled

Logical interface ge-0/0/0.0 (Index 340) (SNMP ifIndex 543) (Generation 149)
  Flags: Up SNMP-Traps 0x4004000 Encapsulation: ENET2
  Traffic statistics:
   Input  bytes  :                44032
   Output bytes  :                58186
   Input  packets:                  540
   Output packets:                  571
  Local statistics:
   Input  bytes  :                44032
   Output bytes  :                58186
   Input  packets:                  540
   Output packets:                  571
  Transit statistics:
   Input  bytes  :                    0                    0 bps
   Output bytes  :                    0                    0 bps
   Input  packets:                    0                    0 pps
   Output packets:                    0                    0 pps
  Protocol inet, MTU: 1500
 Max nh cache: 75000, New hold nh limit: 75000, Curr nh cnt: 1, Curr new hold cnt: 0, NH drop cnt: 0
 Generation: 162, Route table: 0
   Flags: Sendbcast-pkt-to-re
   Addresses, Flags: Is-Preferred Is-Primary
     Destination: 10.100.12/24, Local: 10.100.12.1, Broadcast: 10.100.12.255, Generation: 141
 Protocol multiservice, MTU: Unlimited, Generation: 163, Route table: 0
   Flags: Is-Primary
   Policer: Input: __default_arp_policer__
```

That's an extensive list of statistics, including counters such as unicast, broadcast, and multicast packets, input, output, and CRC errors, and rejects packets.

TIP If you see a large number of errors on an interface you can suspect some physical layer problems happening on the interface.

TIP If you are looking at this output on a switch, and you see a large number of broadcast packets, for example, you might want to issue the command a few more times. If you notice the number is rapidly incrementing you can suspect a L2 loop somewhere.

In all cases, there is one knob that you will find very useful: `refresh`!!:

```
user_1@mx960> show interfaces ge-0/0/0 extensive | match  Broadcast packets | refresh
---(refreshed at 2020-05-02 02:41:40 UTC)---
    Broadcast packets                       0               0
---(refreshed at 2020-05-02 02:41:45 UTC)---
    Broadcast packets                       0               0
---(refreshed at 2020-05-02 02:41:50 UTC)---
    Broadcast packets                       0               0
---(*more 100%)---[abort]
```

The example here is just to demonstrate the use of this feature. By default the command is issued again every five seconds, but you can adjust the frequency if needed:

```
user_1@mx960> show interfaces ge-0/0/0 extensive | match  Broadcast packets | refresh ?
Possible completions:
  <[Enter]>            Execute this command
  <interval>           Time period between refreshes (1..604800 seconds)
  |                    Pipe through a command
```

You should also keep in mind that your device will start to increment these counters the moment you bring the interface up. Let's say that you installed your router last year. Today you are troubleshooting a problem and when you look at the interface statistics you notice hundreds of packet drops. Your instinct might tell you that this these packet drops are the cause of your problem, but it turns out that this counter has been on for a year. Can you tell if they are being caused by a problem right now? Or whether these drops happened some time ago, or has it just gradually happened since the interface was connected?

As done before, we can enter the command several times or use the refresh option to see if the drops are incrementing, or we can clear the counters and see if new packet drops are being recorded.

To clear the counters use the `clear interface statistics <interface>`. Be aware that this will clear ALL counters. Also, if you do not specify, the interface counters for ALL interfaces will be cleared.

In some cases, you might need to look at traffic statistics in real time. You might need, for example, to determine if traffic is leaving the correct interface or if you are receiving traffic from the neighboring router. You might also want to check the traffic rate on the interface.

You can see these real time statistics by issuing `the monitor interface <interface>` command. This command will bring up a screen that updates in real-time the specified interface statistics:

```
user_1@mx960> monitor interface ge-0/0/0

vMX1                              Seconds: 7                  Time: 03:13:52
                                                            Delay: 0/0/1

Interface: ge-0/0/0, Enabled, Link is Up
Encapsulation: Ethernet, Speed: 1000mbps
```

```
Traffic statistics:                                          Current delta
  Input bytes:                 17385772 (757976 bps)              [598836]
  Output bytes:                17438944 (798760 bps)              [609104]
  Input packets:                  12850 (65 pps)                     [414]
  Output packets:                 12976 (69 pps)                     [423]
Error statistics:
  Input errors:                       0                              [0]
  Input drops:                        0                              [0]
  Input framing errors:               0                              [0]
  Carrier transitions:                3                              [0]
  Output errors:                      0                              [0]
  Output drops:                       0                              [0]

Next='n', Quit='q' or ESC, Freeze='f', Thaw='t', Clear='c', Interface='i'
```

You can move to the next interface with `n`, freeze/thaw the output with `f` or 't, clear the counters with c, or move to a specific interface with I, and then specify the interface as shown below:

```
vMX1                              Seconds: 7                 Time: 03:13:52
                                                           Delay: 0/0/1
Interface: ge-0/0/0, Enabled, Link is Up
Encapsulation: Ethernet, Speed: 1000mbps
Traffic statistics:                                          Current delta
  Input bytes:                 17385772 (757976 bps)              [598836]
  Output bytes:                17438944 (798760 bps)              [609104]
  Input packets:                  12850 (65 pps)                     [414]
  Output packets:                 12976 (69 pps)                     [423]
Error statistics:
  Input errors:                       0                              [0]
  Input drops:                        0                              [0]
  Input framing errors:               0                              [0]
  Carrier transitions:                3                              [0]
  Output errors:                      0                              [0]
  Output drops:                       0                              [0]

New interface: ge-0/0/3
Next='n', Quit='q' or ESC, Freeze='f', Thaw='t', Clear='c', Interface='i'


vMX1                              Seconds: 2                 Time: 03:17:36
                                                           Delay: 0/0/1
Interface: ge-0/0/3, Enabled, Link is Up
Encapsulation: Ethernet, Speed: 1000mbps
Traffic statistics:                                          Current delta
  Input bytes:                 39571780 (797792 bps)              [203040]
  Output bytes:                39624354 (797992 bps)              [203025]
  Input packets:                  28186 (69 pps)                     [141]
  Output packets:                 28401 (68 pps)                     [141]
Error statistics:
  Input errors:                       0                              [0]
  Input drops:                        0                              [0]
  Input framing errors:               0                              [0]
  Carrier transitions:                3                              [0]
  Output errors:                      0                              [0]
  Output drops:                       0                              [0]

Next='n', Quit='q' or ESC, Freeze='f', Thaw='t', Clear='c', Interface='i'
```

To exit out of this mode, you just enter `q` or `ESC`.

If you want to see the traffic statistics for multiple interfaces at the same time you can issue the `monitor interface traffic` command. This command will bring up a screen that updates the statistics in real-time of ALL interfaces. To exit out of this mode, you just enter the letter `q`:

```
user_1@mx960> monitor interface ge-0/0/0
```

```
vMX1                            Seconds: 16                 Time: 03:24:19

Interface    Link  Input packets        (pps)     Output packets        (pps)
 ge-0/0/0      Up          42344         (68)             42658         (68)
 lc-0/0/0      Up              0                              0
 pfh-0/0/0     Up              0                              0
 ge-0/0/1      Up            939          (0)              1256          (0)
 ge-0/0/2      Up            945          (0)              1262          (0)
 ge-0/0/3      Up          42348         (68)             42690         (68)
 ge-0/0/4    Down              0          (0)                 0          (0)
 ge-0/0/5    Down              0          (0)                 0          (0)
 ge-0/0/6    Down              0          (0)                 0          (0)
 ge-0/0/7    Down              0          (0)                 0          (0)
 demux0        Up              0                              0
 dsc           Up              0                              0
 em1           Up              0                              0
 esi           Up              0                              0
 fti0          Up              0                              0
 fti1          Up              0                              0
 fti2          Up              0                              0
 fti3          Up              0                              0
 fti4          Up              0                              0
 fti5          Up              0                              0
 fti6          Up              0                              0
 fti7          Up              0                              0
 fxp0          Up              0                              0
 gre           Up              0                              0
 ipip          Up              0                              0
 irb           Up              0                              0
 jsrv          Up              0                              0
 lo0           Up          10798                          10798
 lsi           Up              0                              0
 mif           Up              0                              0
 mtun          Up              0                              0
 pimd          Up              0                              0
 pime          Up              0                              0
 pip0          Up              0                              0
 pp0           Up              0                              0
 rbeb          Up              0                              0
 tap           Up              0                              0
 vtep          Up              0                              0


Bytes=b, Clear=c, Delta=d, Packets=p, Quit=q or ESC, Rate=r, Up=^U, Down=^D
```

You can change the counters from packet per seconds to bits per seconds by pressing `b`, and scroll up and down the interface list with `^U`, and `^D`:

```
vMX1                            Seconds: 110                Time: 03:25:53
```

```
Interface    Link     Input bytes        (bps)      Output bytes        (bps)
 ge-0/0/0      Up        69492948     (797872)          69557452     (798128)
 lc-0/0/0      Up               0                             0
 pfh-0/0/0     Up               0                             0
 ge-0/0/1      Up           76452          (0)            141117          (0)
 ge-0/0/2      Up           76568          (0)            141291        (632)
 ge-0/0/3      Up        69492944     (797872)          69560418     (797872)
 ge-0/0/4    Down               0          (0)                 0          (0)
 ge-0/0/5    Down               0          (0)                 0          (0)
 ge-0/0/6    Down               0          (0)                 0          (0)
 ge-0/0/7    Down               0          (0)                 0          (0)
 demux0        Up               0                             0
 dsc           Up               0                             0
 em1           Up               0                             0
 esi           Up               0                             0
 fti0          Up               0                             0
 fti1          Up               0                             0
 fti2          Up               0                             0
 fti3          Up               0                             0
 fti4          Up               0                             0
 fti5          Up               0                             0
 fti6          Up               0                             0
 fti7          Up               0                             0
 fxp0          Up               0                             0
 gre           Up               0                             0
 ipip          Up               0                             0
 irb           Up               0                             0
 jsrv          Up               0                             0
 lo0           Up        23024617                      23024617
 lsi           Up               0                             0
 mif           Up               0                             0
 mtun          Up               0                             0
 pimd          Up               0                             0
 pime          Up               0                             0
 pip0          Up               0                             0
 pp0           Up               0                             0
 rbeb          Up               0                             0
 tap           Up               0                             0
 vtep          Up               0                             0

Bytes=b, Clear=c, Delta=d, Packets=p, Quit=q or ESC, Rate=r, Up=^U, Down=^D
```

Again, to exit out of this mode, you just enter the letter `q` or `ESC`.

So far we have talked about statistics for the interfaces: both a quick snapshot, and in real-time:

`show interface [extensive]` = snapshot of the interfaces status, attributes, and counters
`monitor interface traffic` = real time traffic stats for all interfaces
`monitor interface <interface-name>` = real time traffic stats for the specified interfaces

But if we want to look at the traffic on that particular interface, we'll cover packet capture next.

Packet Capture

When troubleshooting certain issues it is often helpful to be able to display packets

entering or exiting a Junos device. However, you need to remember the two types of traffic that we discussed at the beginning of the chapter: transit versus exception traffic (user traffic versus control traffic).

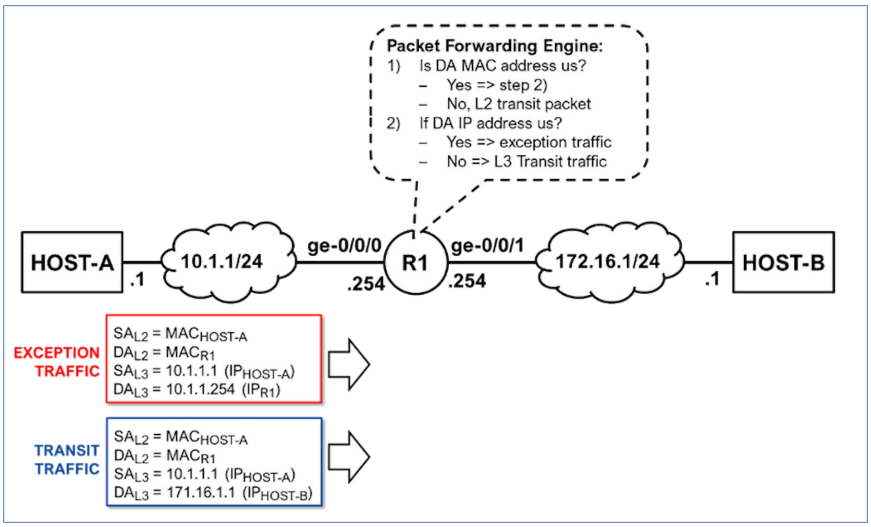


Figure 4.3 Transit Versus Exception Traffic

Most of the time when you are troubleshooting an issue on your device you will be checking exception traffic, in other words traffic going to the routing engine or from the routing engine, such as OSPF updates, ICMP packets, NTP, and so on.

Junos allows you to capture this traffic through the use of the `monitor traffic [<interface>]` command, which gives you access to the tcpdump tool from the CLI. When you run the command, capturing will begin immediately, and will be stopped only when you enter `^c`.

Again, this tool monitors traffic that originates or terminates on the routing engine, in other words, you can only capture traffic destined to or originated by your device. You will *not* see any traffic that is being forwarded by the device (traversing through it). To capture transit traffic, you would need something like port mirroring or sampling, which is not covered in this book, but is addressed in other *Day One* books on the Junos CLI.

When you enter the `monitor traffic` command, if you do not specify an interface, the management interface of your device (therefore: fxp0, me0, em0, etc. depending on your platform) will be monitored as is shown in the next example:

```
root@vMX1> monitor traffic
[edit system]
root@vMX1# run monitor traffic
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
```

```
Address resolution timeout is 4s.
Listening on fxp0, capture size 96 bytes

Reverse lookup for 100.123.1.0 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

21:38:47.944697 Out IP truncated-ip - 124 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 2263545363:2263545507(144) ack 115368021 win 32850
21:38:47.944736 Out IP truncated-ip - 124 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 144:288(144) ack 1 win 32850
21:38:47.944780 Out IP truncated-ip - 76 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 288:384(96) ack 1 win 32850
21:38:47.944817 Out IP truncated-ip - 76 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 384:480(96) ack 1 win 32850
21:38:48.026392  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 480 win 509
21:38:48.944968 Out IP truncated-ip - 10 bytes missing! 100.123.1.0.55496 > dns.google.
domain:  13719+[|domain]
21:38:48.996291  In IP truncated-ip - 67 bytes missing! dns.google.
domain > 100.123.1.0.55496:  13719 NXDomain[|domain]
21:38:48.996464 Out IP truncated-ip - 13 bytes missing! 100.123.1.0.63680 > dns.google.
domain:  64435+[|domain]
21:38:48.996557 Out IP truncated-ip - 220 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 480:720(240) ack 1 win 32850
21:38:49.007452  In IP truncated-ip - 70 bytes missing! dns.google.
domain > 100.123.1.0.63680:  64435[|domain]
21:38:49.007562 Out IP truncated-ip - 796 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 720:1536(816) ack 1 win 32850
21:38:49.143857  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 1536 win 513
21:38:49.898192 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
21:38:50.009746 Out IP truncated-ip - 6 bytes missing! 100.123.1.0.49273 > dns.google.
domain:  10652+[|domain]
21:38:50.236413  In IP truncated-ip - 30 bytes missing! dns.google.
domain > 100.123.1.0.49273:  10652[|domain]
21:38:50.236679 Out IP truncated-ip - 11 bytes missing! 100.123.1.0.56392 > dns.google.
domain:  13481+[|domain]
21:38:50.236764 Out IP truncated-ip - 924 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 1536:2480(944) ack 1 win 32850
21:38:50.251306  In IP truncated-ip - 39 bytes missing! dns.google.
domain > 100.123.1.0.56392:  13481[|domain]
21:38:50.251379 Out IP truncated-ip - 156 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 2480:2656(176) ack 1 win 32850
21:38:50.320447  In IP truncated-ip - 140 bytes missing! pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: P 1:161(160) ack 2480 win 509
21:38:50.320575 Out IP truncated-ip - 28 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 2656:2704(48) ack 161 win 32850
21:38:50.369703  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 2656 win 508
21:38:50.573258  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 2704 win 508
21:38:51.254784 Out IP truncated-ip - 1340 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 2704:4064(1360) ack 161 win 32850
21:38:51.600449  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 4064 win 513
21:38:51.884626 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
```

```
ntp: NTPv4, Client, length 48
21:38:52.254717 Out IP truncated-ip - 428 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 4064:4512(448) ack 161 win 32850
21:38:52.487321  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 4512 win 511
21:38:53.254698 Out IP truncated-ip - 316 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 4512:4848(336) ack 161 win 32850
21:38:53.540319  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 4848 win 510
21:38:53.884705 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
21:38:54.254714 Out IP truncated-ip - 428 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 4848:5296(448) ack 161 win 32850
21:38:54.387002  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 5296 win 508
21:38:55.254857 Out IP truncated-ip - 316 bytes missing! 100.123.1.0.ssh > pool-96-231-209-121.
washdc.fios.verizon.net.57017: P 5296:5632(336) ack 161 win 32850
21:38:55.519409  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 5632 win 513
21:38:55.884631 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
^C
40 packets received by filter
0 packets dropped by kernel
```

This output includes all exception traffic in and out of the fxp0 interface. Being that this is the management interface, and no transit traffic will be forwarded if it arrived on this interface, it is unlikely any traffic other than control traffic will be present.

You can see that at the moment there is SSH traffic, which is the session we are currently using to communicate with the router, as well as some NTP traffic.

You probably also noticed that some entries display a URL, for example `pool-96-231-209-121.washdc.fios.verizon.net` or `ntp-b.nist.gov.ntp`, instead of the IP address – and sometimes we network engineers and administrators want to see those IP addresses that we are familiar with instead.

What if I'm only interested in looking at the NTP traffic? What if I need more details than just the source and destination IP address and the port numbers? These are different options you can specify depending on what you want to capture and the level of detail that you need:

```
root@vMX1> monitor traffic ?
Possible completions:
  <[Enter]>            Execute this command
  absolute-sequence    Display absolute TCP sequence numbers
  brief                Display brief output
  count                Number of packets to receive (0..1000000 packets)
  detail               Display detailed output
  extensive            Display extensive output
  interface            Name of interface
  layer2-headers       Display link-level header on each dump line
  matching             Expression for headers of receive packets to match
```

```
no-domain-names      Don't display domain portion of hostnames
no-promiscuous       Don't put interface into promiscuous mode
no-resolve           Don't attempt to print addresses symbolically
no-timestamp         Don't print timestamp on each dump line
print-ascii          Display packets in ASCII when displaying in hexadecimal format
print-hex            Display packets in hexadecimal format
read-file            Read packets from a given file
resolve-timeout      Period of time to wait for each name resolution (1..4294967295 seconds)
size                 Amount of each packet to receive (bytes)
write-file           Write packets to specified file
|                    Pipe through a command
```

NOTE This list was taken from a vMX running 19.4R1.10. Some of these options might not be available, or might be hidden (will not show when you enter the question mark), if you are running an older version of Junos. Example: `write-filter` and `read-file`.

Let's explore some of these options now, beginning with the most complicated but perhaps most useful: *how to filter the output*. In our example, we're only interested in looking at the NTP messages. But we only had a few NTP packets intermixed with a significant amount of SSH packets. In the real world, there will be even more clutter around our NTP packets. Thus, we need to be able to filter out all those extra packets or it's going to be a long and tedious process, and you just might actually miss the packet that you need!

Junos allows you to pass filters to the underlying tcpdump application from the command line, using the `matching regular expressions` option.

NOTE The `| match` option that we have described before for filtering show commands, is not going to help you in this case. You need to use `matching regular expressions` option.

The filter can be as simple as `matching udp` or as complex as `matching (src || dst 192.168.1.1)&&tcp&&port 22`. Quotes are required when your filter contains two or more words. Here are some examples of valid filters:

- `matching icmp`
- `matching udp`
- `matching tcp`
- `matching host 10.1.1. 2`
- `matching udp port 53`
- `matching proto ospf`
- `matching tcp&&port 22`
- `matching src || dst 192.168.1.1`
- `matching (src || dst 192.168.1.1)&&tcp&&port 22`

Let's get back to our query: *How do we capture only NTP packets?*

In this particular case, you can simply use `monitor traffic matching udp`, since there is only SSH and NTP traffic on the fxp0 interface, as seen when we first captured traffic:

```
root@vMX1> monitor traffic matching udp
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on fxp0, capture size 96 bytes

Reverse lookup for 100.123.1.0 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

21:58:28.101916 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
21:58:28.240032 Out IP truncated-ip - 10 bytes missing! 100.123.1.0.61268 > dns.google.
domain:  52575+[|domain]
21:58:28.252303  In IP truncated-ip - 67 bytes missing! dns.google.
domain > 100.123.1.0.61268:  52575 NXDomain[|domain]
21:58:28.252472 Out IP truncated-ip - 11 bytes missing! 100.123.1.0.55559 > dns.google.
domain:  29618+[|domain]
21:58:28.265333  In IP truncated-ip - 39 bytes missing! dns.google.domain > 100.123.1.0.55559:  ---
more---
^C
132 packets received by filter
0 packets dropped by kernel
```

The filter helped, but if we had other UDP traffic, we would not have had the full benefit of using a filter. Let's be more specific and add the port number to the filter:

```
root@vMX1> monitor traffic matching udp&&port ntp
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on fxp0, capture size 96 bytes

Reverse lookup for 100.123.1.0 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

22:01:34.875676 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
22:01:35.715857 Out IP truncated-ip - 10 bytes missing! 100.123.1.0.57441 > dns.google.
domain:  53353+[|domain]
22:01:35.843987  In IP truncated-ip - 67 bytes missing! dns.google.
domain > 100.123.1.0.57441:  53353 NXDomain[|domain]
22:01:35.844331 Out IP truncated-ip - 11 bytes missing! 100.123.1.0.61886 > dns.google.
domain:  1925+[|domain]
22:01:35.856893  In IP truncated-ip - 39 bytes missing! dns.google.
domain > 100.123.1.0.61886:  1925[|domain]
22:01:36.865688 Out IP truncated-ip - 6 bytes missing! 100.123.1.0.64880 > dns.google.
domain:  20249+[|domain]
22:01:36.868978  In IP truncated-ip - 30 bytes missing! dns.google.
domain > 100.123.1.0.64880:  20249[|domain]
```

```
22:01:36.875580 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
22:01:38.893729 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
^C
144 packets received by filter
0 packets dropped by kernel
```

The `&&` represents a `logical AND`, and in this case, it means that packets need to match protocol UDP `AND` port number NTP.

A `logical OR` is represented with `| |`. If for example, you want to search for packets with either the source address *OR* the destination equal 192.168.1.1, you can enter:

```
root@vMX1> monitor traffic matching src||dst 192.168.1.1
```

MORE? You can find more details about how to write the expressions, the different operators, and some additional examples, at Juniper's TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/monitor-traffic.html.

As recommended for the `| match` option, practice using these matching filters every chance you get so that you can become proficient in using them; they will save you *a lot of time* when verifying protocol operations, and of course, when troubleshooting.

Now what if we want to monitor an interface other than the management interface (e.g. fxp0)? Simple, just add that interface in your command. The next example captures traffic on interface ge-0/0/2 and filters the output so that only OSPF packets are captured:

```
root@vMX1# run monitor traffic interface ge-0/0/2 matching proto 89
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on ge-0/0/3, capture size 96 bytes

Reverse lookup for 10.100.15.2 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

22:37:42.418318  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:37:44.583541 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:37:51.963257  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:37:53.223388 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
^C
4 packets received by filter
0 packets dropped by kernel
```

You can enter the name of the protocol instead of the number for many standard protocols as shown here with OSPF:

```
root@vMX1# run monitor traffic interface ge-0/0/2 matching proto ospf
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on ge-0/0/3, capture size 96 bytes

Reverse lookup for 10.100.15.1 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

22:38:20.231658 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:38:27.811509  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:38:28.715675 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:38:35.778383  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:38:38.074185 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
^C
5 packets received by filter
0 packets dropped by kernel
```

Well, you now know how to capture OSPF packets, but what if you see this:

```
root@vMX1# run monitor traffic interface ge-0/0/3 matching proto ospf
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on ge-0/0/3, capture size 96 bytes

Reverse lookup for 10.100.15.2 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

22:40:55.940423  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:40:56.296151 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:05.297584  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:05.395606 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:05.992011 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 60
22:41:06.994549  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, LS-Ack, length 44
22:41:09.047254 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 60
22:41:10.003574  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, LS-Update, length 60
22:41:11.005489 Out IP truncated-ip - 4 bytes missing! 10.100.15.1 > ospf-all.mcast.net: OSPFv2, LS-
Ack, length 44
22:41:12.858456  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:12.991170  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:12.991969  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:13.008903 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:13.037875  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 56
22:41:13.038390  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 56
```

```
22:41:13.039214 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:13.039538 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:13.044618  In IP 10.100.15.2 > 10.100.15.1: OSPFv2, Database Description, length 44
22:41:13.044623  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:13.044941  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:13.045640 Out IP truncated-
ip - 4 bytes missing! 10.100.15.1 > 10.100.15.2: OSPFv2, Database Description, length 44
22:41:13.046001 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
22:41:13.048030  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
22:41:13.085638 Out IP truncated-
ip - 344 bytes missing! 10.100.15.1 > 10.100.15.2: OSPFv2, Database Description, length 384
22:41:13.125278 Out IP truncated-ip - 156 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 196
22:41:13.212794 Out IP truncated-ip - 136 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 176
22:41:13.248902 Out IP truncated-ip - 60 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 100
22:41:13.287840 Out IP truncated-ip - 92 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 132
22:41:17.443446  In IP 10.100.15.2 > 10.100.15.1: OSPFv2, Database Description, length 44
22:41:17.443721 Out IP truncated-
ip - 344 bytes missing! 10.100.15.1 > 10.100.15.2: OSPFv2, Database Description, length 384
22:41:17.483255  In IP 10.100.15.2 > 10.100.15.1: OSPFv2, Database Description, length 364
22:41:17.484151 Out IP truncated-
ip - 4 bytes missing! 10.100.15.1 > 10.100.15.2: OSPFv2, Database Description, length 44
22:41:17.534250 Out IP truncated-ip - 72 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Update, length 112
22:41:17.536055  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, LS-Update, length 132
22:41:18.537774 Out IP truncated-ip - 24 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, LS-Ack, length 64
22:41:18.539773  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, LS-Ack, length 44
22:41:19.091778 Out IP truncated-ip - 168 bytes missing! 10.100.15.1 > 10.100.15.2: OSPFv2, LS-
Update, length 208
22:41:19.093921  In IP 10.100.15.2 > 10.100.15.1: OSPFv2, LS-Ack, length 64
22:41:21.139453 Out IP truncated-ip - 20 bytes missing! 10.100.15.1 > ospf-all.mcast.
net: OSPFv2, Hello, length 60
^C
40 packets received by filter
0 packets dropped by kernel
```

There was an OSPF Link State Update coming in from your neighbor 10.100.13.2, but what was updated? This is to show you that sometimes, in order to actually get meaningful information, you have to include the `detail` or even the `extensive` option with your monitor command.

Let's go back in time. Yes, we can do that, too – going to the neighbor and using `rollback 1/commit` a couple of times, and including the extensive option when we capture the traffic:

```
root@vMX1> monitor traffic interface ge-0/0/2 matching proto ospf extensive
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on ge-0/0/2, capture size 1514 bytes
```

```
22:45:02.138361  In
        Juniper PCAP Flags [Ext, no-L2, In], PCAP Extension(s) total length 16
          Device Media Type Extension TLV #3, length 1, value: Ethernet (1)
          Logical Interface Encapsulation Extension TLV #6, length 1, value: Ethernet (14)
          Device Interface Index Extension TLV #1, length 2, value: 152
          Logical Interface Index Extension TLV #4, length 4, value: 343
        -----original packet-----
      PFE proto 2 (ipv4): (tos 0xc0, ttl   1, id 49610, offset 0, flags [none], proto: OSPF (89),
length: 76) 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 56 [len 44]
      Router-ID 10.101.101.5, Backbone Area, Authentication Type: none (0)
      Options [External, LLS]
        Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 128
        Designated Router 10.100.15.2
        LLS: checksum: 0xfff6, length: 3
          Extended Options (1), length: 4
            Options: 0x00000001 [LSDB resync]
22:45:02.140035 Out
      Juniper PCAP Flags [Ext], PCAP Extension(s) total length 16
        Device Media Type Extension TLV #3, length 1, value: Ethernet (1)
        Logical Interface Encapsulation Extension TLV #6, length 1, value: Ethernet (14)
        Device Interface Index Extension TLV #1, length 2, value: 152
        Logical Interface Index Extension TLV #4, length 4, value: 343
      -----original packet-----
      00:50:56:a2:3b:39 > 01:00:5e:00:00:05, ethertype IPv4 (0x0800), length 94: (tos 0xc0, ttl   1,
id 52373, offset 0, flags [none], proto: OSPF (89), length: 80) 10.100.15.1 > ospf-all.mcast.net:
OSPFv2, Hello, length 60 [len 48]
      Router-ID 10.100.100.1, Backbone Area, Authentication Type: none (0)
      Options [External, LLS]
        Hello Timer 10s, Dead Timer 40s, Mask 255.255.255.0, Priority 128
        Designated Router 10.100.15.2, Backup Designated Router 10.100.15.1
        Neighbor List:
          10.101.101.5
        LLS: checksum: 0xfff6, length: 3
          Extended Options (1), length: 4
            Options: 0x00000001 [LSDB resync]
---more---
22:45:06.725566  In
      Juniper PCAP Flags [Ext, no-L2, In], PCAP Extension(s) total length 16
        Device Media Type Extension TLV #3, length 1, value: Ethernet (1)
        Logical Interface Encapsulation Extension TLV #6, length 1, value: Ethernet (14)
        Device Interface Index Extension TLV #1, length 2, value: 152
        Logical Interface Index Extension TLV #4, length 4, value: 343
      -----original packet-----
      PFE proto 2 (ipv4): (tos 0xc0, ttl   1, id 49834, offset 0, flags [none], proto: OSPF (89),
length: 152) 10.100.15.2 > ospf-all.mcast.net: OSPFv2, LS-Update, length 132
      Router-ID 10.101.101.5, Backbone Area, Authentication Type: none (0), 2 LSAs
        LSA #1
        Advertising Router 10.101.101.5, seq 0x80000015, age 1s, length 52
          Router LSA (1), LSA-ID: 10.101.101.5
          Options: [External, Demand Circuit]
          Router LSA Options: [none]
            Neighbor Network-ID: 10.100.15.2, Interface Address: 10.100.15.2
              topology default (0), metric 1
            Neighbor Network-ID: 10.100.35.2, Interface Address: 10.100.35.2
                 topology default (0), metric 1
              Stub Network: 10.101.101.5, Mask: 255.255.255.255
                topology default (0), metric 0
              Stub Network: 172.16.100.5, Mask: 255.255.255.255
```

```
                topology default (0), metric 0
              0x0000: 0000 0004 0a64 0f02 0a64 0f02 0200 0001
              0x000f: 0a64 2302 0a64 2302 0200 0001 0a65 6505
              0x001f: ffff ffff 0300 0000 ac64 6405 ffff ffff
              0x002f: 0300 0000
            LSA #2
            Advertising Router 10.101.101.5, seq 0x80000001, age 1s, length 12
              Network LSA (2), LSA-ID: 10.100.35.2
              Options: [External, Demand Circuit]
              Mask 255.255.255.0
              Connected Routers:
                10.101.101.5
                10.100.100.3
              0x0000: ffff ff00 0a65 6505 0a64 6403
---more---
^C
24 packets received by filter
0 packets dropped by kernel
```

Initially, we saw hello messages in and out, and then we saw the update coming in. Understanding all the OSPF details in the output is outside the scope this book, but while you may not be able tell what the update was about before, when you add the `extensive` option you can actually see the contents of the packet, and what your neighbor is updating. In the case of the hello packets, you can see OSPF information like the router ID, or the area ID.

Now, let's look at some of the packets that we have captured so far in the different examples:

```
21:38:53.540319  In IP pool-96-231-209-121.washdc.fios.verizon.
net.57017 > 100.123.1.0.ssh: . ack 4848 win 510

22:01:34.875676 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48

22:41:13.044623  In IP 10.100.15.2 > ospf-all.mcast.net: OSPFv2, Hello, length 60
```

They all have something in common: some of the IP addresses and port numbers have been replaced with names. While sometimes these names make perfect sense, many times we are so used to the addresses of some of our devices, for example, that we would rather see the IP addresses displayed.

Let's look at the NTP for example:

```
root@vMX1> monitor traffic matching udp&&port ntp
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on fxp0, capture size 96 bytes

Reverse lookup for 100.123.1.0 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

22:59:49.829048 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48
```

```
22:59:51.835813 Out IP truncated-ip - 16 bytes missing! 100.123.1.0.ntp > ntp-b.nist.gov.
ntp: NTPv4, Client, length 48

^C
194 packets received by filter
0 packets dropped by kernel
```

The address of the NTP server is being translated to `ntp-b.nist.gov.ntp`, and instead of displaying the port number it shows `.ntp`. If we want to see the IP address instead, add `no-resolve` to the command:

```
root@vMX1> monitor traffic matching udp&&port ntp no-resolve
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on fxp0, capture size 96 bytes

23:02:07.245707 Out IP truncated-
ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
23:02:09.246390 Out IP truncated-
ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
23:02:11.259337 Out IP truncated-
ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
23:02:13.245848 Out IP truncated-
ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
^C
102 packets received by filter
0 packets dropped by kernel
```

Now you can see the addresses and the port number for NTP 123, as we wanted.

The next option looks at `layer2-headers`, which will help you when you need to know the source and destination MAC addresses of packets. Here is the NTP example again, just for comparison:

```
root@vMX1> monitor traffic matching udp&&port ntp no-resolve layer2-headers
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is OFF.
Listening on fxp0, capture size 96 bytes

23:06:14.172105 Out 00:50:56:01:01:00 > 00:50:56:a2:8f:f5, ethertype IPv4 (0x0800), length 74:
truncated-ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
23:06:16.172047 Out 00:50:56:01:01:00 > 00:50:56:a2:8f:f5, ethertype IPv4 (0x0800), length 74:
truncated-ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
23:06:18.179278 Out 00:50:56:01:01:00 > 00:50:56:a2:8f:f5, ethertype IPv4 (0x0800), length 74:
truncated-ip - 16 bytes missing! 100.123.1.0.123 > 132.163.96.5.123: NTPv4, Client, length 48
^C
107 packets received by filter
0 packets dropped by kernel
```

After looking at all these examples, we hope you realize how useful this command is for verification and troubleshooting tasks. Many times you will be looking at these packet captures right on your desktop, so you should find it useful that you can save your packet captures to a standard pcap file that you can then use for offline analysis with third-party tools such as Wireshark. To do this, add the `write-file` option and specify a filename for the capture:

```
root@vMX1> monitor traffic matching udp&&port ntp no-resolve layer2-headers write-file dns.pcap
```

NOTE Remember that this option might be hidden depending on the version of Junos you are running.

If you do not specify a directory, the file will be saved in your home directory as described earlier in the chapter.

It is highly recommended that you limit the number of packets captured using this method so that the .pcap file does not fill the entire flash drive of the device. Add the `count` option to specify the number of packets to capture:

```
root@vMX1> monitor traffic matching udp&&port ntp no-resolve layer2-headers write-file dns.
pcap count 150
```

To finish this section, let's summarize the commands one last time, as we know they can get mixed-up in your brain. Pay attention to the command syntax:

`show interface [extensive]` = snapshot of the interfaces status, attributes, and counters

`monitor interface traffic` = real time traffic stats for all interfaces

`monitor interface <interface-name>` = real time traffic stats for the specified interfaces

`monitor traffic interface <interface-name>` = actual packet capturing on specified interface. If interface not specified, packet captured on management interface (e.g. fxp0)

Basic Networking Tools

There are some common CLI tools that are used for troubleshooting, accessing devices, and even testing to validate connectivity. They include SSH, telnet, ping, and traceroute.

The most basic troubleshooting that you can do from a device is called *Packet Internet Gopher* more commonly known as ping. Ping will send an ICMP ECHO_REQUEST packet to a remote IP address that will, if alive, respond with an ECHO_REPLY packet.

In Junos there are a few knobs and attributes that you can use to customize your ping. For example, if you just enter `ping <destination-address>`, Junos will continue sending ECHO_REQUEST packets, until you stop it with a break (CTRL-C):

```
user@vSRX1-JNCIA> ping 192.168.255.2
PING 192.168.255.2 (192.168.255.2): 56 data bytes
64 bytes from 192.168.255.2: icmp_seq=0 ttl=64 time=1.205 ms
64 bytes from 192.168.255.2: icmp_seq=1 ttl=64 time=0.913 ms
```

```
64 bytes from 192.168.255.2: icmp_seq=2 ttl=64 time=0.986 ms
64 bytes from 192.168.255.2: icmp_seq=3 ttl=64 time=0.927 ms
64 bytes from 192.168.255.2: icmp_seq=4 ttl=64 time=0.791 ms
^C
--- 192.168.255.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.791/0.964/1.205/0.136 ms
```

For each ping the device reports the response time, and after the ping has stopped, reports the results in the form of number of packets transmitted, number of packet received, loss percentage, and also the `round-trip min/avg/max/stddev`. This can be helpful as a first indication of possible congestion somewhere in the path.

If you append `rapid` to the end of the ping command, the device will send five packets and then stop. A request will be sent as soon as a reply is received without delay, thus the keyword `rapid`:

```
user@vSRX1-JNCIA> ping 192.168.255.2 rapid
PING 192.168.255.2 (192.168.255.2): 56 data bytes
!!!!!
--- 192.168.255.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.525/0.736/1.117/0.202 ms
```

There are many other options:

```
{master}
user_1@mx960> ping ?
Possible completions:
  <host>               Hostname or IP address of remote host
  count                Number of ping requests to send (1..2000000000 packets)
  do-not-fragment      Don't fragment echo request packets (IPv4)
  inet                 Force ping to IPv4 destination
  inet6                Force ping to IPv6 destination
  interface            Source interface (multicast, all-ones, unrouted packets)
  interval             Delay between ping requests (seconds)
  logical-system       Name of logical system
  routing-instance     Routing instance for ping attempt
  size                 Size of request packets (0..65468 bytes)
  source               Source address of echo request
  tos                  IP type-of-service value (0..255)
  ttl                  IP time-to-live value (IPv6 hop-limit value) (hops)
  verbose              Display detailed output
[…]
```

You can change the interval and the size of the packets so that you can have a particular packet rate, for example:

```
user_1@mx960> ping 10.10.1.2 interval 0.1 size 1250
PING 10.10.1.2(10.10.1.2): 1250 data bytes
1258 bytes from 10.10.1.2: icmp_seq=0 ttl=64 time=0.889 ms
1258 bytes from 10.10.1.2: icmp_seq=1 ttl=64 time=0.930 ms
1258 bytes from 10.10.1.2: icmp_seq=2 ttl=64 time=0.916 ms
1258 bytes from 10.10.1.2: icmp_seq=3 ttl=64 time=1.177 ms
1258 bytes from 10.10.1.2: icmp_seq=4 ttl=64 time=0.960 ms
1258 bytes from 10.10.1.2: icmp_seq=5 ttl=64 time=0.915 ms
```

```
1258 bytes from 10.10.1.2: icmp_seq=6 ttl=64 time=32.551 ms
1258 bytes from 10.10.1.2: icmp_seq=7 ttl=64 time=0.948 ms
1258 bytes from 10.10.1.2: icmp_seq=8 ttl=64 time=0.942 ms
^C
--- 10.10.1.2 ping statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.889/4.470/32.551/9.929 ms
```

You can see we're sending 1250 bytes every 0.1 sec, which gives a rate of 100000bps. Now let's change the size and also set the `do-not-fragment` bit to test the MTU:

```
user_1@mx960> ping 10.10.1.2 interval 0.1 size 5000 do-not-fragment
PING 10.10.1.2 (10.10.1.2): 5000 data bytes
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
ping: sendto: Message too long
^C^
--- 10.10.1.2 ping statistics ---
8 packets transmitted, 0 packets received, 100% packet loss
```

Now, by default, when you ping a destination, the source address of the echo request message is the address of the interface used to send the packets. In Figure 4.4, when I ping 10.10.1.2, the source address will be 10.1.1.1:

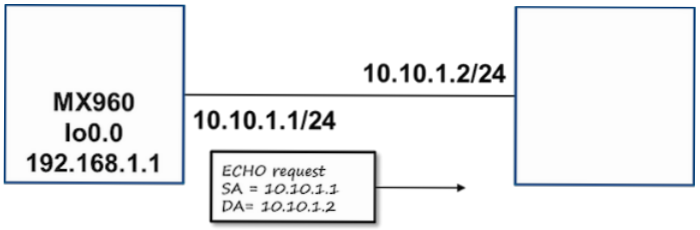


Figure 4.4 *Source Address*

What if I want to ping from the loopback to validate that the destination knows how to send traffic back to my loopback address? See Figure 4.5.

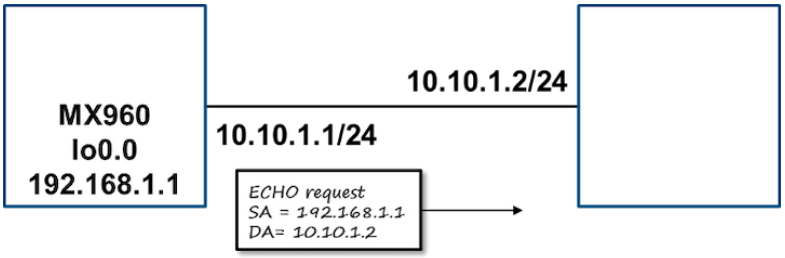


Figure 4.5 Destination Address

We can specify the source address of the packets like this:

```
{master}
user_1@mx960> ping 10.10.1.2 interval 0.1 size 1250 source 192.168.1.1
PING 10.10.1.2 (10.10.1.2 ): 1250 data bytes
1258 bytes from 10.10.1.2: icmp_seq=0 ttl=64 time=1.079 ms
1258 bytes from 10.10.1.2: icmp_seq=1 ttl=64 time=0.934 ms
1258 bytes from 10.10.1.2: icmp_seq=2 ttl=64 time=0.960 ms
1258 bytes from 10.10.1.2: icmp_seq=3 ttl=64 time=0.953 ms
1258 bytes from 10.10.1.2: icmp_seq=4 ttl=64 time=0.937 ms
1258 bytes from 10.10.1.2: icmp_seq=5 ttl=64 time=0.963 ms
1258 bytes from 10.10.1.2: icmp_seq=6 ttl=64 time=1.145 ms
1258 bytes from 10.10.1.2: icmp_seq=7 ttl=64 time=0.939 ms
1258 bytes from 10.10.1.2: icmp_seq=8 ttl=64 time=0.959 ms
1258 bytes from 10.10.1.2: icmp_seq=9 ttl=64 time=0.955 ms
1258 bytes from 10.10.1.2: icmp_seq=10 ttl=64 time=0.947 ms
1258 bytes from 10.10.1.2: icmp_seq=11 ttl=64 time=0.970 ms
^C
--- 10.10.1.2  ping statistics ---
12 packets transmitted, 12 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.934/0.978/1.145/0.062 ms
```

Also, by default, when I ping an address the device performs a route lookup in inet.0 to figure out which interface to send the packet out of, and who the next hop is.

What if my interface belongs to a routing instance? This is shown in Figure 4.6.

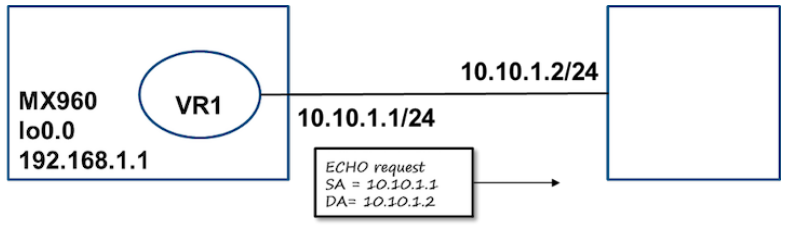


Figure 4.6 Routing Instance

You can add the `routing-instance` information to the command so that the router knows the route lookup has to be performed in that routing table. (in the example, that would be VR1.inet.0):

```
user_1@mx960> ping 10.10.1.2 interval 0.1 size 1250 routing-instance VR1
PING 10.10.1.2 (10.10.1.2): 1250 data bytes
1258 bytes from 10.10.1.2: icmp_seq=0 ttl=64 time=0.959 ms
1258 bytes from 10.10.1.2: icmp_seq=1 ttl=64 time=0.945 ms
1258 bytes from 10.10.1.2: icmp_seq=2 ttl=64 time=0.960 ms
1258 bytes from 10.10.1.2: icmp_seq=3 ttl=64 time=0.938 ms
1258 bytes from 10.10.1.2: icmp_seq=4 ttl=64 time=0.943 ms
1258 bytes from 10.10.1.2: icmp_seq=5 ttl=64 time=1.200 ms
1258 bytes from 10.10.1.2: icmp_seq=6 ttl=64 time=0.962 ms
1258 bytes from 10.10.1.2: icmp_seq=7 ttl=64 time=0.953 ms
1258 bytes from 10.10.1.2: icmp_seq=8 ttl=64 time=0.958 ms
^C
--- 10.10.1.2  ping statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.938/0.980/1.200/0.078 ms
```

And what if I want to use ping to quickly test CoS classification? This is shown in Figure 4.7.

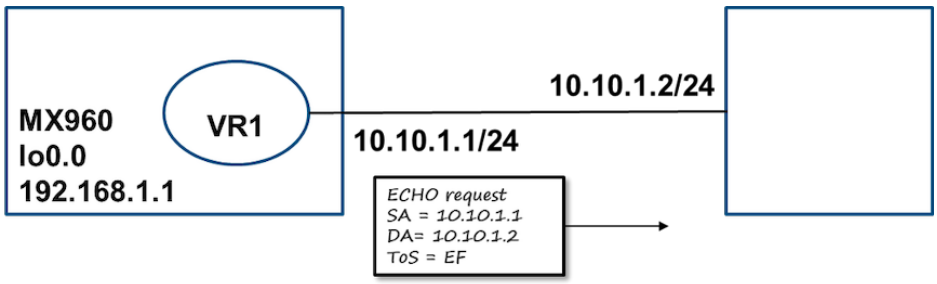


Figure 4.7 CoS Classification

```
user_1@mx960> ping 10.10.1.2 interval 0.1 size 1250 source 137.39.4.17 tos 184 detail
PING 10.10.1.2 (10.10.1.2): 1250 data bytes
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=0 ttl=64 time=0.981 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=1 ttl=64 time=0.955 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=2 ttl=64 time=0.979 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=3 ttl=64 time=0.948 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=4 ttl=64 time=0.956 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=5 ttl=64 time=0.961 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=6 ttl=64 time=0.961 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=7 ttl=64 time=1.467 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=8 ttl=64 time=0.966 ms
1258 bytes from 10.10.1.2 via ge-0/0/0.0: icmp_seq=9 ttl=64 time=7.919 ms
^C
--- 10.10.1.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.948/1.709/7.919/2.075 ms
```

This example also shows you the `option` detail, which includes the outbound interface for the echo request packet.

The next tool to describe is `traceroute`.

Figuring out the path that a packet is taking can also be important in troubleshooting. We have initiated a traceroute to a remote IP address of 192.168.254.1 and you can see next that it was two hops away. Our first hop was 192.168.0.2 and the second hop was 192.168.254.1:

```
user@vSRX1-JNCIA> traceroute 192.168.254.1
traceroute to 192.168.254.1 (192.168.254.1), 30 hops max, 52 byte packets
 1  192.168.0.2 (192.168.0.2)  1.058 ms  0.839 ms  0.564 ms
 2  192.168.254.1 (192.168.254.1)  9.186 ms  10.281 ms  9.988 ms
```

In the next example, we are doing a traceroute to 192.168.254.2, but we never received a reply back. This means that either the device is down, or the device is not responding to ICMP ECHO_REQUESTS (it might have a firewall filter or does not have a route back to the source of the requests). This is also a typical response when there is a firewall in the path blocking the ICMP messages:

```
user@vSRX1-JNCIA> traceroute 192.168.254.1
traceroute to 192.168.254.1 (192.168.254.1), 30 hops max, 52 byte packets
 1  192.168.0.2 (192.168.0.2)  1.294 ms  0.749 ms  0.551 ms
 2  * * *
 3  * * *
^C
```

There is also a traceroute option that can be used to monitor in real-time. It is based off a utility called *My traceroute* and is invoked from the CLI by issuing the `traceroute monitor <destination>` command. This command displays the live traceroute as well as reports back on loss, packets sent/received, response, etc.:

```
user@vSRX1-JNCIA> traceroute monitor 192.168.254.1

                             My traceroute  [v0.69]
vSRX1-JNCIA (0.0.0.0)(tos=0x0 psize=64 bitpattern=0x00)Fri Apr 24 20:00:36 2020
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                      Packets               Pings
 Host                                Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 192.168.0.2                       0.0%     8    1.0   0.8   0.7   1.0   0.1
 2. 192.168.253.1                     0.0%     7    9.9   9.8   6.4  12.7   2.4
 3. 192.168.254.1                     0.0%     7    9.3   8.0   6.3  10.5   1.5
```

The last tool we will describe in this section is `telnet`, which can be used to access a remote device and also to check for connectivity to a remote device on a particular port number. While most of us know that this is used to access and manage devices, did you know that you can also test connectivity to things like web servers? This is a handy way to be sure that you can actually access a remote device that you may not have direct access to, and that there is no firewall or filter in the path blocking that particular port number.

To test with `telnet`, all you need to do is specify the port that you want to connect to the remote server. In the next example, we are attempting to establish a connection using port 80. As you can see in the output, we connect to 192.168.254.1 successfully:

```
user@vSRX1-JNCIA> telnet 192.168.254.1 port 80
Trying 192.168.254.1...
Connected to 192.168.254.1.
Escape character is '^]'.
^C
```

What happens if the remote server is not listening on that port? In the next example we tried to access the same server, but this time on port 81. As demonstrated, the connection was refused:

```
user@vSRX1-JNCIA> telnet 192.168.254.1 port 81
Trying 192.168.254.1...
telnet: connect to address 192.168.254.1: Connection refused
telnet: Unable to connect to remote host
```

Also, know that `telnet` and `traceroute` also have similar options to ping:

```
ylara@MX960-LR16-VX2.LAX7-re0> telnet ?
Possible completions:
  <host>               Hostname or address or remote host
  port                 Port number or service name on remote host
  routing-instance     Name of routing instance for telnet session
  source               Source address to use in telnet connection
[…]


ylara@MX960-LR16-VX2.LAX7-re0> traceroute ?
Possible completions:
  <host>               Hostname or address of remote host
  routing-instance     Name of routing instance for traceroute attempt
  source               Source address to use in outgoing traceroute packets
  tos                  IP type-of-service field (IPv4) (0..255)
[…]
```

Realtime Performance Monitoring

The tools we have discussed so far are a reactive way to troubleshoot an issue. As most of us know in networking, we usually find out about an issue after service is down and degraded and people are complaining.

Junos offers a way to monitor *reachability* and keep a record of it so that you can have a baseline and you can detect deviations from that baseline to help catch the problem before it becomes an issue with complaining customers and coworkers!

It is called an *RPM* for real-time performance monitoring (it's SRX specific). While the configuration of RPM and all its options is beyond the scope of this book, here's the configuration for a simple ICMP probe called `JNCIA` that pings 192.168.254.1 every 60 seconds, to give you an idea of what RPM can do for you:

```
user@vSRX1-JNCIA> show configuration services rpm
probe JNCIA {
    test ICMP {
        target address 192.168.254.1;
        probe-count 5;
        probe-interval 60;
    }
}
```

To view the results of a probe, issue the `show service rpm probe-results` command. In the output from that command you can see that the owner is `JNCIA`, the test is ICMP, and we have a target address of 192.168.254.1:

```
user@vSRX1-JNCIA> show services rpm probe-results
    Owner: JNCIA, Test: ICMP
    Target address: 192.168.254.1, Probe type: icmp-ping, Icmp-id: 11,
    Test size: 5 probes
    Probe results:
      Response received
      Probe sent time: Fri Apr 24 21:11:45 2020
      Probe rcvd/timeout time: Fri Apr 24 21:11:45 2020, No hardware timestamps
      Rtt: 2206 usec, Round trip jitter: -2579 usec
      Round trip interarrival jitter: 1596 usec
    Results over current test:
      Probes sent: 4, Probes received: 4, Loss percentage: 0.000000
      Measurement: Round trip time
        Samples: 4, Minimum: 2206 usec, Maximum: 4785 usec, Average: 3484 usec,
        Peak to peak: 2579 usec, Stddev: 914 usec, Sum: 13936 usec
      Measurement: Positive round trip jitter
        Samples: 1, Minimum: 1391 usec, Maximum: 1391 usec, Average: 1391 usec,
        Peak to peak: 0 usec, Stddev: 0 usec, Sum: 1391 usec
      Measurement: Negative round trip jitter
        Samples: 3, Minimum: 157 usec, Maximum: 2579 usec, Average: 1313 usec,
        Peak to peak: 2422 usec, Stddev: 992 usec, Sum: 3938 usec
   [ Output omitted ]
```

To view a cleaner history of an RPM probe, issue the `show services rpm history-results brief owner JNCIA` command. You can see that we had a few timeouts during the test that were caused by us shutting down the interface so that we would see the errors:

```
user@vSRX1-JNCIA> show services rpm history-results brief owner JNCIA
    Owner, Test  Probe Sent               Probe received            Round trip time
    JNCIA, ICMP  Fri Apr 24 20:40:38 2020 Fri Apr 24 20:40:38 2020          5940 usec
    JNCIA, ICMP  Fri Apr 24 20:59:42 2020 Fri Apr 24 20:59:42 2020          6146 usec
    JNCIA, ICMP  Fri Apr 24 21:00:42 2020 Fri Apr 24 21:00:42 2020          3698 usec
    JNCIA, ICMP  Fri Apr 24 21:00:43 2020 Fri Apr 24 21:00:43 2020          2417 usec
    JNCIA, ICMP  Fri Apr 24 21:01:43 2020 Fri Apr 24 21:02:43 2020 Request timed out
    JNCIA, ICMP  Fri Apr 24 21:02:43 2020 Fri Apr 24 21:03:43 2020 Request timed out
    JNCIA, ICMP  Fri Apr 24 21:03:43 2020 Fri Apr 24 21:03:43 2020          6067 usec
    JNCIA, ICMP  Fri Apr 24 21:04:43 2020 Fri Apr 24 21:04:43 2020          4244 usec
    JNCIA, ICMP  Fri Apr 24 21:07:44 2020 Fri Apr 24 21:07:44 2020          2068 usec
```

Junos OS Installation

From time to time you may need to perform a software upgrade on your Juniper device to gain access to a new feature or implement a fix for a software bug, and it's important to understand the naming convention of the software releases.

Junos uses the following convention:

Package – Release – Edition

The package is the name of the Junos OS package. Depending on the hardware or software platform you are running, the package name will be different.

The following table shows a number of different package naming conventions across various Juniper hardware platforms:

Table 4.2 Naming Conventions

Package Name	Platform
jinstall-ex	EX (2200, 3200, 3300, 4200, 4300, 4500 series)
junos-arm	EX (2300, 3400 series)
junos-srxsme	SRX (100, 200, 300, 500, 600 series)
junos-srxentedge	SRX (1500 series)
junos-srx1k3k	SRX (1400, 3000 series)
junos-vsrx	VSRX
jinstall-ppc	MX (5/10/40/80, 100 series)
junos-vmhost-install-mx-x86-64	MX (10K)
vmx-bundle	vMX
jinstall-host-qfx-5	QFX (5000 series)
jinstall-vqfx	VQFX

MORE? Details about the different available packages can be found here: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/junos-os-installation-packages.html.

Junos software releases are delivered for all platforms on a (roughly) quarterly basis, with the first number being the year of release and the second being the quarter of the year. Examples:

19.1 was released in March 2019

19.2 was released in June 2019

19.3 was released in September 2019

19.4 was released in December 2019

20.1 was released in March 2020

Major releases such as these will generally include new features and/or support for new hardware platforms as they are released.

Junos OS release numbering is displayed in the following format:

Release Number – Release Type – Build Number – Spin Number

Some current release versions include 20.1R1.11, 19.4R1.10, and 18.3R3.8.

The R in these examples, represents a *revenue* release, meaning a release for customer deployment. The first revenue release is R1. Releases from R2 onward are considered maintenance releases.

The numbers after the R, are the build and spin numbers. These are generally bug fixes and quality improvements to existing major releases. The number and cadence of these releases will vary between products and code versions, but in general the higher the build number, the more tested the release of code is.

IMPORTANT To find the JTAC suggested version for your device, be sure to visit Juniper support at: https://kb.juniper.net/InfoCenter/index?page=content&id=kb21476.

Prior to upgrading any Junos device, be sure to check on available free space on the device, to make sure there is sufficient space for upgrading. To check the storage, issue the `show system storage` command. You want to check for the free space where the `/var/tmp` mapping is – here it's `/dev/da0s3d` and it shows free space of `339M`:

```
user@EX2200-C> show system storage
fpc0:
--------------------------------------------------------------------------
Filesystem             Size       Used      Avail  Capacity   Mounted on
/dev/da0s1a            183M       112M        56M       66%  /
devfs                  1.0K       1.0K         0B      100%  /dev
/dev/md0                58M        58M         0B      100%
[--- OUTPUT OMITTED ---]
/dev/da0s3e            123M       2.5M       111M        2%  /var
/dev/md9               167M        12K       154M        0%  /tmp
/dev/da0s3d            369M        26K      339M         0%  /var/tmp
/dev/da0s4d             62M       130K        57M        0%  /config
```

If you need to free up space, issue the `request system storage cleanup` command, which provides a list of unnecessary files as well as old log files and the CLI will ask you whether you want to delete them. You can run the command with the `no-confirm` option, which will instruct the router to go ahead and remove the files without user confirmation.

To see what would be deleted, before deleting anything, you can append `dry-run` to the command:

```
user@EX2200-C> request system storage cleanup dry-run
fpc0:
--------------------------------------------------------------------------

List of files to delete:

         Size Date         Name
    11B Jan 21 02:08 /var/jail/tmp/alarmd.ts
   143B Jan 21 02:08 /var/log/default-log-messages.0.gz
  11.1K Jan 21 02:08 /var/log/messages.0.gz
    60B Jan 21 01:52 /var/log/wtmp.0.gz
    57B Jan 20 07:04 /var/log/wtmp.1.gz
 124.0K Jan 20 07:04 /var/tmp/gres-tp/env.dat
     0B Jan 20 07:04 /var/tmp/gres-tp/lock
     0B Jan 20 07:04 /var/tmp/rtsdb/if-rtsdb
```

To upgrade a Junos device, first download the appropriate image from Juniper.net and then copy the image to the device. You can accomplish this by transferring the file via SCP, TFTP, FTP, USB, and even directly from Juniper via HTTPS. This file is normally copied to the `/var/tmp` directory on the device.

Once you have the image copied to your device, issue the `request system software add /var/tmp/[image name] reboot` command.

Here we upgrade an EX2200-C that is running 12.3R3.4 to 12.3R12.4, and then have the system reboot after installation. The keyword `reboot` at the end of the install string indicates that action. If you do not specify the reboot keyword, the system will activate the software the next time it reboots:

```
{master:0}
user@EX2200-C> show version
fpc0:
--------------------------------------------------------------------------
Hostname: EX2200-C
Model: ex2200-c-12p-2g
JUNOS Base OS boot [12.3R3.4]
JUNOS Base OS Software Suite [12.3R3.4]
JUNOS Kernel Software Suite [12.3R3.4]
JUNOS Crypto Software Suite [12.3R3.4]
JUNOS Online Documentation [12.3R3.4]
JUNOS Enterprise Software Suite [12.3R3.4]
JUNOS Packet Forwarding Engine Enterprise Software Suite [12.3R3.4]
JUNOS Routing Software Suite [12.3R3.4]
JUNOS Web Management [12.3R3.4]
JUNOS FIPS mode utilities [12.3R3.4]

{master:0}
user@EX2200-C> request system software add /var/tmp/jinstall-ex-2200-12.3R12.4-domestic-signed.
tgz reboot

Installing disk0s3d:/jinstall-ex-2200-12.3R12.4-domestic-signed.tgz
Verified jinstall-ex-2200-12.3R12.4-domestic.tgz signed by PackageProduction_12_3_0
date: connect: Can't assign requested address
```

```
Checking package integrity...
Running requirements check first for jbundle-ex-2200-12.3R12.4-...
Running pre-install for jbundle-ex-2200-12.3R12.4-...
Installing jbundle-ex-2200-12.3R12.4- in /tmp/installer.tmp/pa2899.11/jbundle-ex-2200-12.3R12.4-
domestic.x2899...
Running post-install for jbundle-ex-2200-12.3R12.4-...
Verified SHA1 checksum of fips-mode-arm-12.3R12.4.tgz
Verified SHA1 checksum of jbase-ex-12.3R12.4.tgz
Verified SHA1 checksum of jboot-ex-12.3R12.4.tgz
Verified SHA1 checksum of jcrypto-ex-12.3R12.4.tgz
```

Once the system has rebooted, you can log in and verify that the code has been upgraded by issuing the `show version` command. Here you can see that we went from version 12.3R3.4 to 12.3R12.4:

```
user@EX2200-C> show version
fpc0:
--------------------------------------------------------------------------
Hostname: EX2200-C
Model: ex2200-c-12p-2g
JUNOS Base OS boot [12.3R12.4]
JUNOS Base OS Software Suite [12.3R12.4]
JUNOS Kernel Software Suite [12.3R12.4]
JUNOS Crypto Software Suite [12.3R12.4]
JUNOS Online Documentation [12.3R12.4]
JUNOS Enterprise Software Suite [12.3R12.4]
JUNOS Packet Forwarding Engine Enterprise Software Suite [12.3R12.4]
JUNOS Routing Software Suite [12.3R12.4]
JUNOS Web Management [12.3R12.4]
JUNOS FIPS mode utilities [12.3R12.4]
```

If you are running a device that has multiple routing engines (REs), you can upgrade the software by performing an in-service software upgrade, better known as *ISSU*. This allows you to upgrade with no disruption on the control plane and minimal disruption of traffic. A requirement for this is that you have graceful Routing Engine switchover (GRES) and nonstop active routing (NSR) enabled on your device.

To perform the ISSU upgrade, issue the `request system software in-service-upgrade` command. GRES is enabled with the `set chassis redundancy graceful-switchover` and NSR with `set routing-options nonstop-routing`.

You can validate that NSR is working with `show task replication`:

```
user_1@mx960> show task replication
        Stateful Replication: Enabled
        RE mode: Master

    Protocol                Synchronization Status
    BGP                     Complete
    IS-IS                   Complete
    MPLS                    Complete
    RSVP                    Complete
    LDP                     Complete
```

MORE? For a complete reference guide on how to do software installation and upgrades check out the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/software-installation-and-upgrade/software-installation-and-upgrade.html.

Powering On, Shutting Down, and Rebooting Junos Devices

As mentioned in Chapter 1, Junos OS runs on top of FreeBSD or Linux. As the underlying operating system has some files permanently open (for example, log files), it is important to perform a shutdown before removing the power supply or unplugging the power cable. This ensures that any open files are saved to the flash memory.

If a device running Junos OS is not shut down properly, files could become corrupted or lost. Additionally, the next time the device boots, Junos will know that it was not shut down cleanly and will perform a series of filesystem checks that may significantly increase boot time.

In Junos, there are usually two ways to power down the system depending on the hardware platform: halt the system or power off.

On some devices, you can `halt` the system, which will stop all processes on the box (including the forwarding of traffic) but leave the box powered on and still reachable via the serial console. To halt your system, issue the `request system halt` command. The device will now gracefully stop all the services on the device and wait for a keypress to reboot:

```
user@EX2200-C> request system halt
warning: This command will halt all the members.
If planning to halt only one member use the member option
Halt the system ? [yes,no] (no) yes

Shutdown at Thu Jan 21 02:16:55 2016.
[pid 2518]

syncing disks... All buffers synced.
Uptime: 19h15m50s

The operating system has halted.
Please press any key to reboot.
```

Some systems also support the `power-off` function which, as the name implies, will go a step further than halt and turn off the device completely, requiring physical intervention to restart. This can be achieved by issuing the `request system power-off` command:

```
user@EX2200-C> request system power-off
warning: This command will halt all the members.
If planning to halt only one member use the member option
Power Off the system ? [yes,no] (no) yes


*** System shutdown message from user@EX2200-C ***
```

To reboot the system, you simply issue the `request system reboot` command. You can request the reboot to be performed immediately by just pressing Enter, or you can schedule for a later time. Also, be aware that when you are on a router with redundant REs, issuing the command without specifying which routing engine you want to reboot, the router will reboot the master routing engine, which is the one you would be interacting with when issuing the command. You can tell the router you want the other routing engine to be rebooted (backup), or that you want to reboot both:

```
{master}[edit]
ylara@mx960-lr17-re0> request system reboot ?
Possible completions:
  <[Enter]>            Execute this command
  at                   Time at which to perform the operation
  both-routing-engines   Reboot both the Routing Engines
  in                   Number of minutes to delay before operation
  junos                Boot off Junos volume
  message              Message to display to all users
  oam                  Boot off OAM volume
  other-routing-engine   Reboot the other Routing Engine
  |                    Pipe through a command
```

To schedule a reboot include the option `at` as shown in this example:

```
{master}
user_1@mx960> request system reboot at 21:00
Reboot the system at 21:00? [yes,no] (no) yes

Shutdown at Mon Apr 27 21:00:00 2020.
[pid 91054]

{master}
user_1@mx960> show system reboot
reboot requested by full at Mon Apr 27 21:00:00 2020
[process id 91054]
```

If, later, you want to cancel the reboot you can enter:

```
{master}
user_1@mx960> clear system reboot
Apr 27 15:52:20
reboot requested by full at Mon Apr 27 21:00:00 2020
[process id 91054]
Terminating...
```

Root Password Reset

If you ever forget the root password to a Junos device (or one of your colleagues sets it to something only they know and then goes on vacation for two weeks), you can use the following process to reset it.

First, you must be connected to the device via console when you power it on. During the power up you will need to keep an eye out for the following text:

Press [Enter] to boot immediately, or the spacebar for command prompt.

Do not get distracted, you only have a few seconds to hit the spacebar. As soon as you see this, press the spacebar to access the `loader>` prompt:

```
loader>
```

From the loader prompt, enter the command `boot -s`. (Press the ? key for a list of commands.) This will boot the system into single user mode, which is a special system mode that disables the authentication process within Junos. When prompted, enter `recovery` to go into recovery mode:

```
Enter full pathname of shell or 'recovery' for root password recovery
              or RETURN for /bin/sh: recovery
```

The system will now boot and place you in single user mode. You will receive a banner like the one below and then be placed into > prompt:

```
NOTE: Once in the CLI, you will need to enter configuration mode using
NOTE: the 'configure' command to make any required changes. For example,
NOTE: to reset the root password, type:
NOTE:    configure
NOTE:    set system root-authentication plain-text-password
NOTE:    (enter the new password when asked)
NOTE:    commit
NOTE:    exit
NOTE:    exit
NOTE: When you exit the CLI, you will be asked if you want to reboot
NOTE: the system

Starting CLI ...
warning: This chassis is operating in a non-master role as part of a virtual-chassis (VC) system.
warning: Use of interactive commands should be limited to debugging and VC Port operations.
warning: Full CLI access is provided by the Virtual Chassis Master (VC-M) chassis.
warning: The VC-M can be identified through the show virtual-
chassis status command executed at this console.
warning: Please logout and log into the VC-M to use CLI.
{linecard:0}
root@EX2200-C>
```

Now you can enter configure mode and change the root password with the `set system root-authentication plain-text-password` command:

```
{linecard:0}[edit]
root@EX2200-C# set system root-authentication plain-text-password
New password: juniper123
Retype new password: juniper123

{linecard:0}[edit]
root@EX2200-C# commit and-quit
```

When the configuration has committed, you can reboot the device and log in with your new root password.

Chapter 5

Routing Fundamentals

By Tom Dwyer

In its simplest form, no matter how you pronounce it, routing (ROW-TING/ ROO-TING) is the process of selecting a path to move packets between disparate Layer 3 networks. This chapter will define the different components of routing and the basic requirements for it in Junos. It is truly a first step, and the analogy of putting your big toe into the ocean comes to mind. The key thing to remember is that routing is universal and if you understand the concepts on any device it can only help you to understand it here with Junos. The key is to understand where the knobs are in the configuration and how to turn them. Luckily Juniper does a great job of making this clear, and most of the configuration syntax is organized very neatly into the config within a hierarchy.

Why Do We Route? An Introduction

The purpose of routing is to help to build an end-to-end path between devices across multiple Layer 3 networks. Each Layer 3 router hop along this path must support routing information to allow the forwarding of traffic along the path towards the destination. Each hop along this path may contain a small subset of this routing information such as a single default route. Layer 3 devices along the path might have more complete routing information with hundreds, thousands, and even millions of routes.

Each hop is important. Each hop along this path is a piece of the puzzle. The other thing to remember is that these paths are typically stateless and unidirectional.

What this means to you as an engineer is that the path you take towards the destination is not necessarily the path they would take to you, if they initiated traffic (called *asymmetrical*). The internet is a great example of this as you may take one path out of your network only to have traffic return another path. Some routing decisions will be made locally on the device and kept locally as others will carry weight with it to the overall path. Understanding the impact of this will help you with your choice of tools and how to troubleshoot using those tools. Be forewarned, ping might not always be your friend.

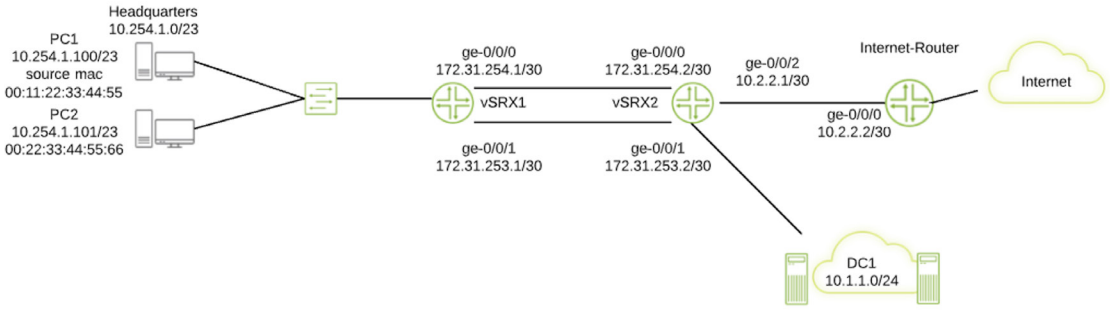


Figure 5.1 *Example Enterprise Network*

In the past we have required dedicated devices called routers to perform this function. This was typically due to the memory required as well as having dedicated processing for the routing of packets. As overall computing resources have increased over the years we have seen this responsibility move to other devices.

- A Layer 3 switch may provide not only a default gateway for your network, it may act as a router to connect to your WAN.
- If you have a requirement for large amounts of routes your network edge might have a dedicated router, or you may have a firewall. (One item to note with the use of a firewall as a router is that firewalls are stateful devices, and with that in mind some of that asymmetric routing path we talked about previously may result in drops.)
- Cloud networks may use a VPN with a routing protocol as the entry point to their network.

While we have new players in the mix the core requirement is still the same: provide a dedicated path to the destination. The destination must be reachable. Each Layer 3 hop you cross along this path has routing information to provide you with the answer to the next piece of the puzzle.

Layer 2 Versus Layer 3

Understanding the basics of Layer 2 versus Layer 3 communication is a great first step. Historically this was broken down based on hardware capabilities. However, in the case of Layer 3 switches, hardware now handles both Layer 2 and Layer 3. These layers work together and are the basic building blocks of connectivity.

Looking at our topology, with our IP address we have the address and then a netmask. One of the purposes of this is to provide the boundaries of a broadcast domain. Within this broadcast domain we communicate directly via Layer 2 as shown in Figure 5.2. When we move away from the broadcast domain we require a device to assist us and that typically is our default gateway.

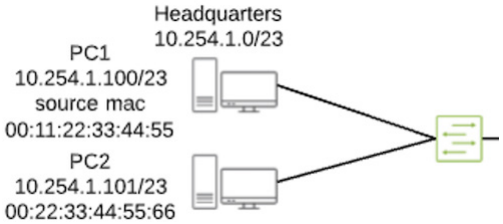


Figure 5.2 *Layer 2 Communications on VLAN 1*

In Figure 5.3 there is a Layer 3 switch with a single Layer 3 interface on it. As we communicate between hosts on the same network we will use Layer 2 communications. So, in the case that PC1 wants to ping PC2 it will need to know the MAC address of PC2 and it initiates an ARP request. Once that is answered PC1 now knows the MAC address of PC2 and can communicate directly. This all happens without a Layer 3 router.

Figure 5.3 *Layer 3 Communications from VLAN 1*

But to connect to something that isn't on our local broadcast domain we will need some help. In this case, let's try to access a server that is on the internet. Now the PC doesn't have the capability for a large routing table. Luckily it has a default gateway. This gateway of last resort will forward packets from PC1 to its next upstream routed hop. We still need some Layer 2 here, and ARP for the MAC address of the default gateway, which is the IRB (integrated routing and bridging) interface on our Layer 3 switch. (44:44:44:44:44:44). Now the Layer 3 switch will take a look in its routing table and see if it knows how to get to the destination. In this case that switch will need to forward the packet to the next hop in the path. This will continue until the packet arrives at its destination or the packet is dropped or rejected.

Route Table

The routing table is where you consolidate the different sources of routing information. Some of this information is learned locally from the interfaces we configure. Some is learned from routing protocols or locally defined resources such as static routes. By default, Junos supports a single active unicast route to each destination for the purposes of forwarding. Junos will take this active route and will use it to construct the forwarding table. The REs' forwarding table then gets installed into the forwarding table on the PFE (packet forwarding engine). Juniper separates the control and forwarding planes (shown in Figure 5.4) as we see this in practice as we move forwarding table entries from the RE to the PFE.

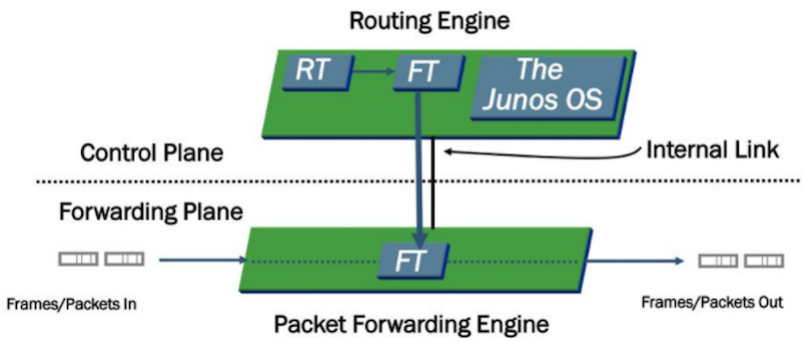


Figure 5.4 *Routing and Forwarding Tables*

One of the main advantages of the approach Juniper takes with its routing tables is how it consolidates families of routes together. So there is a default routing table for IPv4: the *inet.0* routing table. For IPv6 we have the *inet6.0* routing table. This ability to segment the routing table with families of protocols makes it easier to review each family without the noise of other disparate families of data.

Examples of some of the predefined routing tables:

- inet.0 IPv4 unicast routes
- inet.1 used for multicast forwarding
- inet.2 used for storing unicast routes for multicast rpf
- inet.3 used to resolve bgp next-hops for rsvp/ldp label switched path(lsp) endpoints (MPLS)
- inet.4 used for Multicast Source Discovery Protocol (MSDP)
- inet6.0 IPv6 unicast routes
- mpls.0 used to resolve labels to their lsp next-hop (MPLS)

Route Preference

Route preference is a tiebreaker used to determine what route will be installed as the active route when identical length routes are learned from multiple routing protocols and sources. Other vendors may call this *administrative distance*. Route Preference is a locally significant value. It only affects the local routing table with regard to what route gets selected as the active route. Sometimes people conflate route preference with metric. They also may sometimes use metric as a tie-breaking mechanism for static routes. While this may solve a problem in the short term, remember that metric values can affect route selection further in your path as you use them in combination with dynamic routing protocols. When making changes to the behavior of route selection, make sure you validate downstream from your router. Figure 5.5 shows an example of routing preference at work.

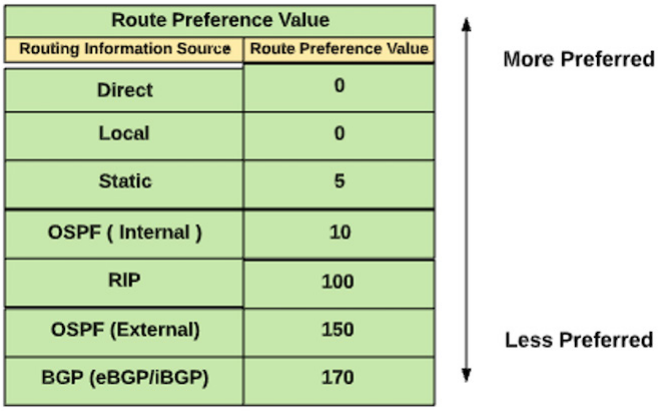


Figure 5.5 Route Preference Metric

You can see that direct and local routes have a route preference of zero. This makes sense as these routes are created when we add the IP addresses on interfaces. Static is a manually defined protocol (see Figure 5.6) and therefore is trusted over routes we hear from other devices.

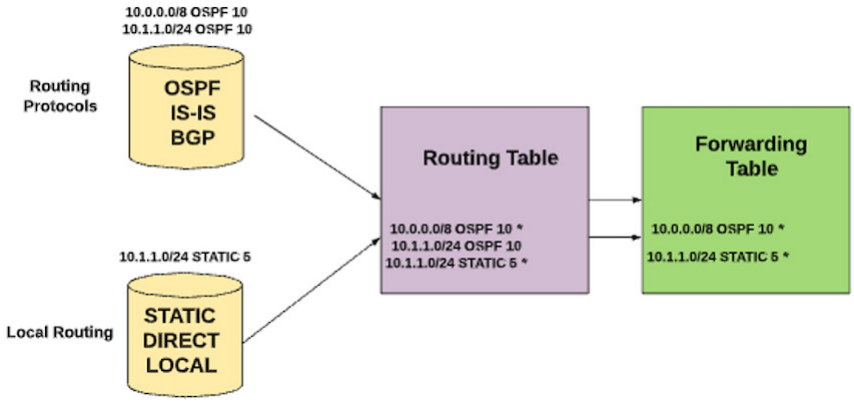


Figure 5.6 Static Routes

Figure 5.7 is an example of the routing table. There is the default inet.0 routing table for IPv4 routes and there are six active routes.

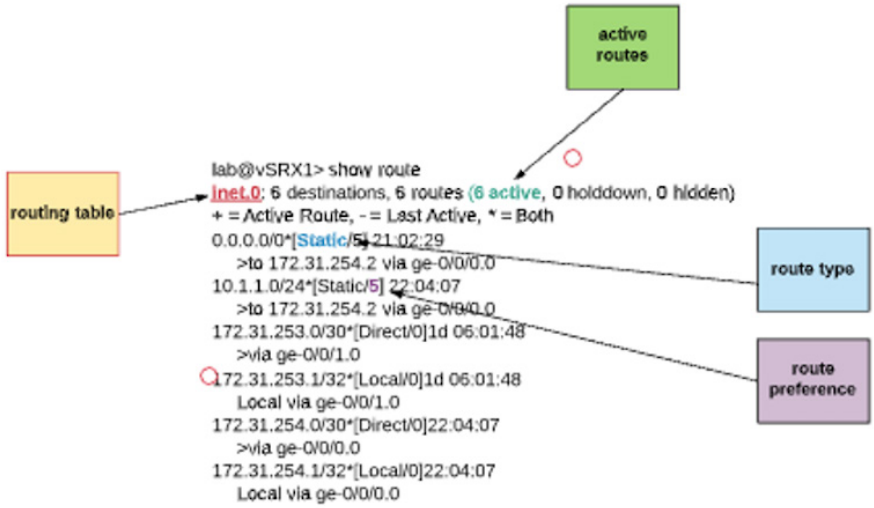


Figure 5.7 Routing Table Example

So, let's build out our routing table. The first step will be to add IPv4 addresses to the interfaces which will create two sets of routes: a direct route with a preference of zero and a local route with a preference of zero. The direct route is created based on the netmask configured in your interface. The local route is the host route for your interface. As these are IPv4 unicast routes, what routing table should they use by default? It's inet.0 of course:

```
lab@vSRX1#set interfaces ge-0/0/0 unit 0 family inet address 172.31.254.1/30
lab@vSRX1#set interfaces ge-0/0/1 unit 0 family inet address 172.31.253.1/30
lab@vSRX1#commit and-quit
```

After you commit, you'll see the interface IP changes installed on the four new routes into the routing table. You have two direct routes based on the netmask defined on the interface and two local routes as well. These routes have a preference of 0 and as you have seen, the lower the number the more likely it is to be installed as an active route. As these are IPv4 unicast routes they will be installed into the inet.0 routing table by default.

Now what happens when you learn routes from multiple protocols? In the next example we are learning a prefix of 10.2.2.0/30 and have three routes in the routing table for it:

```
lab@vSRX1>show route 10.2.2.0/30

inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.2.2.0/30        *[Static/5] 00:00:32
                    >  to 172.31.253.2 via ge-0/0/1.0
                    [OSPF/10] 06:07:50, metric 2
                       to 172.31.254.2 via ge-0/0/0.0
                    >  to 172.31.253.2 via ge-0/0/1.0

Let's take a deeper into why one route was selected over another:
lab@vSRX1>show route 10.2.2.0/30 detail

inet.0: 5 destinations, 6 routes (5 active, 0 holddown, 0 hidden)
10.2.2.0/30 (2 entries, 1 announced)
        *Static Preference: 5
                Next hop: 172.31.253.2 via ge-0/0/1.0, Next hop index: 0
                Address: 0xc29926c
                Next-hop reference count: 2
                State: <Active Int Ext>
                Age: 8:41
                Validation State: unverified
                Task: RT
                Announcement bits (1): 0-KRT
                AS path: I
         OSPF   Preference: 10
                Next hop type: Router, Next hop index: 559
                Address: 0xc299974
                Next-hop reference count: 3
                Next hop: 172.31.253.2 via ge-0/0/1.0, selected
                Session Id: 0x0
                State: <Int>
```

```
Inactive reason: Route Preference
Age: 1:05:27    Metric: 0
Validation State: unverified
        Tag: 0
Task: OSPF
AS path: I
```

You can see that route preference of 5 is lower than 10. So the static route is selected in the routing table as the active route.

The Forwarding Table

The routing table did its job and has selected the active route. So what do you do with that information? How about building a forwarding table? The forwarding table is where data is stored including destination networks, next hops, and egress interfaces for the purpose of forwarding packets (see Figure 5.8). You can examine it by using the operational `show route forwarding-table` command. Here are a couple of things to note: with routes and forwarding the key to remember is the longest prefix match always wins. The more specific the route the higher routing ranks it. So in the case of 192.168.1.0/24 and 192.168.1.0/27, you would see both installed in the routing table, however if you are sending to 192.168.1.5 it will take the next hop associated with the /27, because that is more specific.

In the next output, notice the permanent entry for the default route. It has a reject. If you do not have a default route in your routing table and do not have the destination route you are looking for, you need to generate a response to let the source know that you can't pass the packet past this point. Here we use an ICMP unreachable message that is generated by the route engine. When you configure a default route you'll see a MAC address as the next hop. By default, static routes require a directly connected route, and because it is directly connected, we are using Layer 2 and ARP:

```
lab@vSRX1> show route forwarding-table
Routing table: default.inet
Internet:
Enabled protocols: Bridging,
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            user     0 50:0:0:6:0:1       ucst      575     4 ge-0/0/0.0
default            perm     0                    rjct       36     1
0.0.0.0/32         perm     0                    dscd       34     1
10.1.1.0/24        user     0 172.31.254.2       ucst      575     4 ge-0/0/0.0
172.31.253.0/30    intf     0                    rslv      574     1 ge-0/0/1.0
172.31.253.0/32    dest     0 172.31.253.0       recv      572     1 ge-0/0/1.0
172.31.253.1/32    intf     0 172.31.253.1       locl      573     2
172.31.253.1/32    dest     0 172.31.253.1       locl      573     2
172.31.253.2/32    dest     0 50:0:0:6:0:2       ucst      551     1 ge-0/0/1.0
172.31.253.3/32    dest     0 172.31.253.3       bcst      571     1 ge-0/0/1.0
172.31.254.0/30    intf     0                    rslv      570     1 ge-0/0/0.0
172.31.254.0/32    dest     0 172.31.254.0       recv      564     1 ge-0/0/0.0
172.31.254.1/32    intf     0 172.31.254.1       locl      569     2
172.31.254.1/32    dest     0 172.31.254.1       locl      569     2
```

```
172.31.254.2/32    dest     0 50:0:0:6:0:1       ucst      575     4 ge-0/0/0.0
172.31.254.3/32    dest     0 172.31.254.3       bcst      563     1 ge-0/0/0.0
224.0.0.0/4        perm     0                    mdsc       35     1
224.0.0.1/32       perm     0 224.0.0.1          mcst       31     1
255.255.255.255/32 perm     0                    bcst       32     1
```

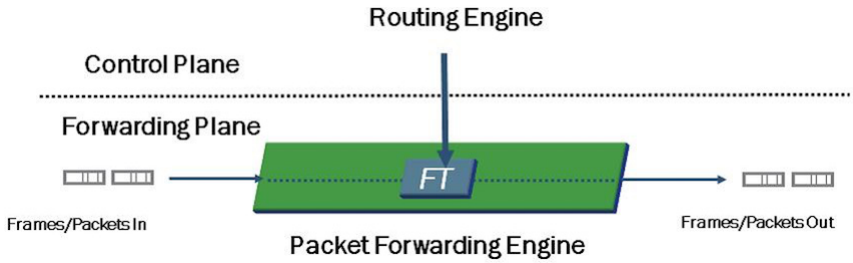


Figure 5.8 Forwarding Table Example

The active routes are now moved to the forwarding table. So in this example we're learning routes from static and OSPF. Two equal length routes for 10.1.1.0/24. The static is the active route per the preference tie-breaker – only that route match for 10.1.1.0/24 is moved to the forwarding table (see Figure 5.9).

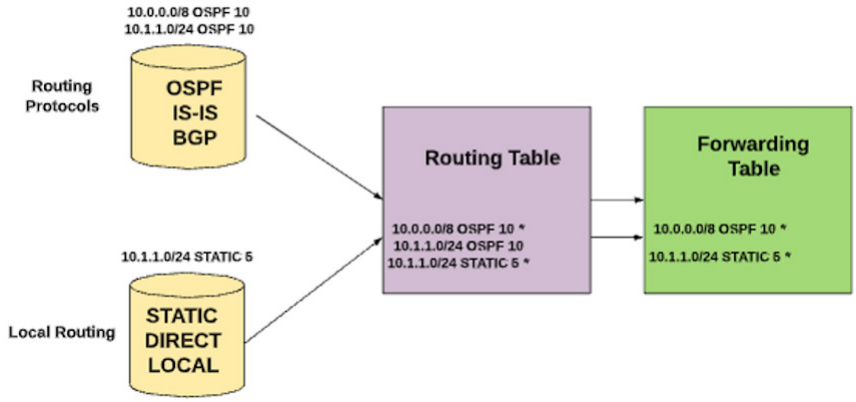


Figure 5.9 Active Route 10.1.1.0/24

The abstraction of the control and forwarding plane has one massive advantage: you can forward packets once the forwarding plane is built using high speed ASICs and dedicated processing. Let's do this via the PFE which is typically an ASIC. The packet arrives on an ingress interface. We perform a destination look-up in the forwarding table. If the packet is destined for a remote network, it is forwarded out the egress next hop interface. Figure 5.10 illustrates different forwarding scenarios and how the forwarding table is built.

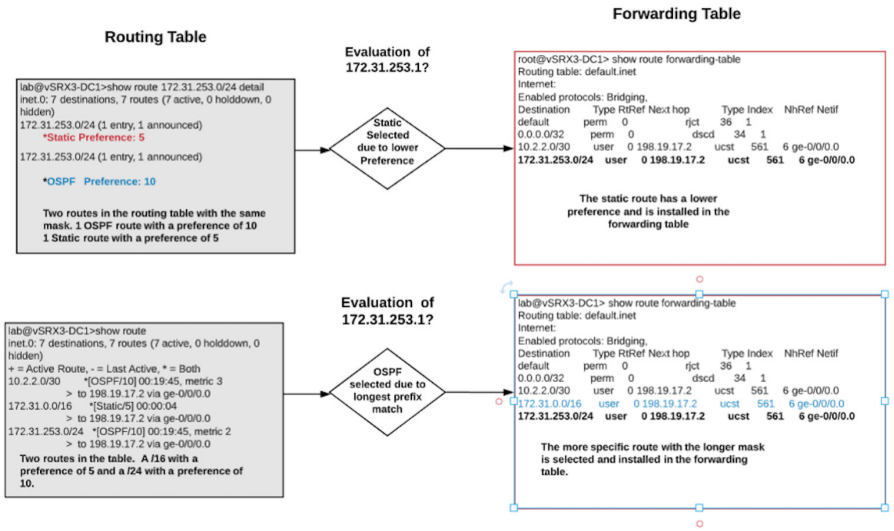


Figure 5.10 Forwarding Scenarios and Forwarding Table

Routing Instances

The default routing instance for Junos is also known as the *master routing instance*, and it contains the default routing tables of inet.0 and inet6.0:

```
lab@vSRX1>show route instance
Instance             Type
        Primary RIB                                  Active/holddown/hidden
master               forwarding
        inet.0                                       6/0/0
        inet6.0                                      1/0/0
```

User defined routing instances can be configured under Junos. The type of routing instance that can be used is based on the model of the Juniper device you are working with. The same physical interface can belong to multiple routing instances as long as the unit number is unique per routing instance:

```
[edit routing-instances jncia-vrouter]      >>> Instance name is a user defined variable.
instance-type virtual-router;               >>> Instance type value
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.5.5.1;
    }
}
protocols {
    ospf {
        area 0.0.0.0 {
            interface irb.10;
            interface lo0.1 {
```

```
                passive;
            }
        }
    }
}
interface irb.10;     >>> Interfaces can only belong to one instance
interface lo0.1;      >>> Only one loopback interface per routing instance.
```

Let's validate that the new routing instance was built. So with the user defined routing instance we now have two tables added, the `jncia-vrouter.inet.0` for unicast IPv4 routes, and `jncia-vrouter.inet6.0` for IPv6 routes:

```
lab@vSRX1>show route instance jncia-vrouter
Instance             Type
        Primary RIB                                  Active/holddown/hidden
jncia-vrouter        virtual-router
        jncia-vrouter.inet.0                         1/0/0
        jncia-vrouter.inet6.0                        1/0/0
```

You need to validate the new routing table. Let's check the IPv4 unicast route table for the custom routing instance:

```
lab@vSRX1>show route table jncia-vrouter.inet.0

jncia-vrouter.inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.1.1/32      *[Static/5] 00:00:03
                       >  to 172.16.1.1 via ge-0/0/0.100
224.0.0.5/32       *[OSPF/10] 05:14:23, metric 1
                       MultiRecv
```

After building a routing instance it's very important to remember how to work with it – you need to remember to reference this as you use operational commands. So, if you wish to test reachability for the `jncia-vrouter routing instance` to 172.16.1.1, you will need to reference the correct routing instance. If you don't it will use the master routing instance and in this case will utilize inet.0 for resolution:

```
lab@vSRX1>ping routing-instance jncia-vrouter 172.16.1.1
PING 172.16.1.1 (172.16.1.1): 56 data bytes
64 bytes from 172.16.1.1: icmp_seq=0 ttl=64 time=2.927 ms
64 bytes from 172.16.1.1: icmp_seq=1 ttl=64 time=0.841 ms
64 bytes from 172.16.1.1: icmp_seq=2 ttl=64 time=0.868 ms
```

Static Routing

Static routes are manually defined routes that you add to each router. They have a preference of 5 by default and require a next hop that is reachable by a directly connected interface by default. If the next hop is not directly connected the route will be hidden. In the next example (Figure 5.11) we are configuring a route and the next hop is 172.31.254.2, which is a directly connected interface.

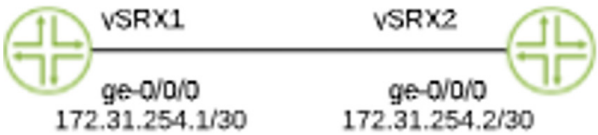


Figure 5.11 Static Routes

```
[edit routing-options]
lab@vSRX1# show
static {
    route 10.1.1.0/24 next-hop 172.31.254.2;
```

This will create a route in the routing table with a preference of 5. This route has been installed as the active route. We can validate this with the `show route protocol static` command – it will only display static routes:

```
lab@vSRX1>show route protocol static

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

10.1.1.0/24        *[Static/5] 1d 04:58:44
                    >  to 172.31.254.2 via ge-0/0/0.0
```

You can also take other actions as part of static routing. Next we're creating a route and setting it to discard. You might want to use this to black-hole traffic. This does that and silently discards the packet to the bit bucket without sending notification:

```
[edit routing-options]
lab@vSRX1# show
static {
    route 192.168.10.0/24 discard
```

You can also drop the packet, and if you wish, send a message back to notify that it is being dropped. The route engine will send an unreachable message with the `reject` knob set:

```
[edit routing-options]
lab@vSRX1# show
static {
    route 192.168.11.0/24 reject;
```

Next, the `no-readvertise` knob prevents this route from being redistributed into another routing protocol. Why would you want to do this? Well, if it is a management network, or security network, you might not want to redistribute it and have an accidental transit of traffic:

```
[edit routing-options]
lab@vSRX1# show
static {
    route 192.168.11.0/24 next-hop 172.31.254.2 no-readvertise;
```

Qualified Next Hops

In the Figure 5.12 example there are two identical routes and you would like to have them installed as a primary and secondary route. The problem with this is that if you use a static route they would both have a preference of 5, creating the need to find a further tie-breaker to install an active route.

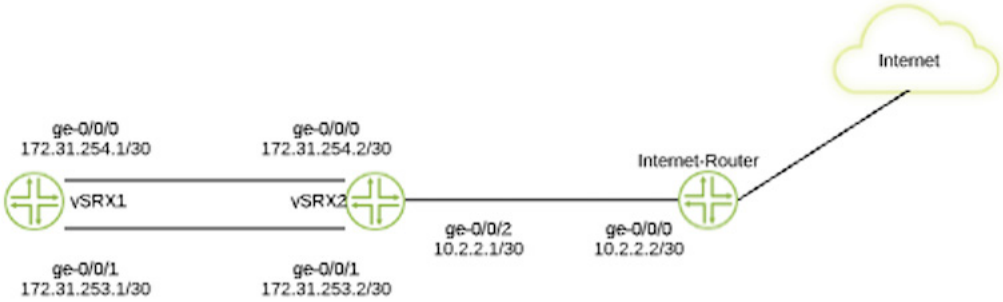


Figure 5.12 Qualified Next-Hop Example

You can add both routes and give them unique preference values creating primary and secondary route values with the `qualified-next-hop` option and also by adding a `preference` value. Now remember, you need to understand the routing protocols that are being used and their preferences: here, we're using OSPF that has a preference of 10 so make sure that your backup route has a value less than that:

```
[edit routing-options]
lab@vSRX1# show
static {
    route 10.1.1.0/24 {
        next-hop 172.31.254.2;
        qualified-next-hop 172.31.253.3 {
            preference 9;
        }
    }
}
```

We installed a primary static route with a next hop of 172.31.254.2. As it is a static route remember it will have a preference of 5. Then we defined a `qualified-next-hop` for 172.31.253.2 and give it a preference of 9. Next, a `show route` command validates that the active route is the static route with the next hop of 172.31.254.2:

```
lab@vSRX1>show route 10.1.1.1

inet.0: 8 destinations, 9 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.0/24        *[Static/5] 00:18:14
                    >  to 172.31.254.2 via ge-0/0/0.0
                    [Static/9] 00:18:14
                    >  to 172.31.253.3 via ge-0/0/1.0

lab@vSRX1>show route 10.1.1.1 detail

inet.0: 8 destinations, 9 routes (8 active, 0 holddown, 0 hidden)
10.1.1.0/24 (2 entries, 1 announced)
        *Static Preference: 5
                Next hop type: Router, Next hop index: 575
                Address: 0xc299a3c
                Next-hop reference count: 4
                Next hop: 172.31.254.2 via ge-0/0/0.0, selected
                Session Id: 0x0
                State: <Active Int Ext>
                Age: 17:13
                Validation State: unverified
                Task: RT
                Announcement bits (2): 0-KRT 2-Resolve tree 1
                AS path: I
         Static Preference: 9
                Next hop type: Router, Next hop index: 0
                Address: 0xc299b04
                Next-hop reference count: 2
                Next hop: 172.31.253.3 via ge-0/0/1.0, selected
                Session Id: 0x0
                State: <Int Ext>
                Inactive reason: Route Preference
                Age: 17:13
                Validation State: unverified
                Task: RT
                AS path: I
```

You can see after adding the `detail` knob to our `show route` command, that the reason the 172.31.253.2 hop is not active is due to a higher route preference of 9 versus 5. Now what happens if we disable the primary path:

```
lab@vSRX1>configure
lab@vSRX1#set interfaces ge-0/0/0 disable
lab@vSRX1# commit
commit complete

[edit]
lab@vSRX1# run show route 10.1.1.1

inet.0: 7 destinations, 8 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.0/24        *[Static/9] 07:55:33
                    >  to 172.31.253.3 via ge-0/0/1.0
```

The `qualified-next-hop` becomes the active route, and when we rollback that configuration the primary static route resumes its role as the active route:

```
{edit}
lab@vSRX1# rollback 1
load complete

[edit]
lab@vSRX1# commit
commit complete

[edit]
lab@vSRX1# run show route 10.1.1.1

inet.0: 8 destinations, 10 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.0/24        *[Static/5] 00:00:05
                    >  to 172.31.254.2 via ge-0/0/0.0
                    [Static/9] 07:57:46
                    >  to 172.31.253.3 via ge-0/0/1.0
```

Static with Resolve Option (Indirect Next Hops)

One of the requirements for defining a static route is that the next hop has to be directly connected. Junos, by default, does not support recursive next hop resolution. This, however, is easily fixed with adding the `resolve` option at the end of the `static route` command. As long as the next hop is resolved by an active route in the routing table or by a default route it will install the resolved route as part of the indirect hop. If you do not add the `resolve` knob the route will remain hidden as there is no valid next hop (see Figure 5.13).

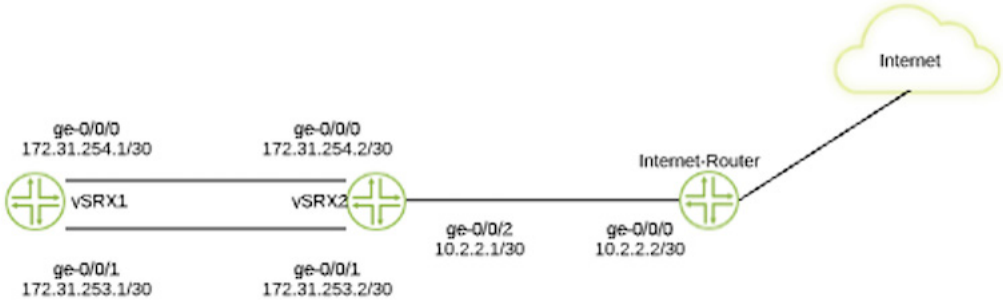


Figure 5.13 Static – Indirect Hop

In Figure 5.13 there are two links between SRX1 and SRX2 for redundancy. The internet router has a connection to SRX2 Here is an example of two hosts on the same VLAN. Both hosts have the same network prefix of 192.168.10 and have the same netmask. In the next example we are going to connect directly via Layer 2. Let's show static routes:

```
[edit routing-options]
lab@vSRX1# show
static {
    route 0.0.0.0/0 {
        next-hop 10.2.2.2;
        resolve;
    }
}
```

The next hop of 10.2.2.2 is in the routing table due to OSPF routes being learned by our neighbor:

```
lab@vSRX1>show route 10.2.2.2

inet.0: 8 destinations, 9 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.2.2.0/30        *[OSPF/10] 00:09:00, metric 2
                       to 172.31.254.2 via ge-0/0/0.0
                    >  to 172.31.253.2 via ge-0/0/1.0
```

Now that we see the route for 10.2.2.2 in the table our next-hop has the ability to resolve. We see that with a check for the default route.

```
lab@vSRX1>show route 0.0.0.0/0

inet.0: 8 destinations, 9 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0.0.0.0/0          *[Static/5] 00:09:03, metric2 2
                    >  to 172.31.254.2 via ge-0/0/0.0
                       to 172.31.253.2 via ge-0/0/1.0
```

Let's validate that this is an indirect or resolved route by looking at the forwarding table. In the next example you can see that this is an indirect route by the `indr` type. The MAC address of the SRX2's ge-0/0/1 interface is the physical next hop:

```
lab@vSRX1>show route forwarding-table
Routing table: default.inet
Internet:
Enabled protocols: Bridging,
Destination        Type RtRef Next hop           Type Index    NhRef Netif
default            user     0                    indr   262143     2
                              50:0:0:6:0:2       ucst      551     4 ge-0/0/1.0
```

Okay, now let's see what is happening in the script.

Dynamic Routing Versus Static Routing

The use of dynamic versus static routing within your network might be a sensitive topic to some readers. Static is very easy to configure, as you have seen in the previous section of this chapter, and it's easy to control as well in a small enterprise network. The scale of the network typically dictates which architecture makes sense for your network. In smaller networks static makes sense. As the network grows, it becomes increasingly more difficult to administer routing across multiple devices with such a manual method. At some point you risk making the architecture too complex.

Let's talk about some of the benefits of dynamic routing. Instead of manually configuring a static route you can have the network learn a path automatically by adding an interface or a local route to the dynamic routing protocol. It learns this and then distributes this across multiple devices, and in some cases can do this at large scale. This allows for diverse paths and redundancy as many routing protocols build or understand alternate redundant paths. Dynamic routing protocols can use some of the tie-breaker mechanisms listed before to find the best path or most optimal route between two networks without the administrator having to manually calculate it. And dynamic routing protocols also allow for the ability to administratively override a path if needed, via cost or routing policy, which is dealt with in Chapter 6.

Dynamic Routing Protocols

Dynamic routing protocols can be broken out into two families: interior gateway protocols (IGPs), or external gateway protocols (EGPs):

IGPs:

- Operate within a single autonomous system
- This allows a single administrative element to provide routing policy and network control
- OSPF, OSPFv3, IS-IS, RIP, RIPNG are examples of IGPs

EGPs:

- Autonomous systems provide a delineation for control and routing policy
- Communication between networks is provided via different administrative controls

BGP is the only EGP in use, as shown in Figure 5.14, with the tiebreaker mechanisms mentioned before to find the best path or most optimal route between two networks without the administrator having to manually calculate this.

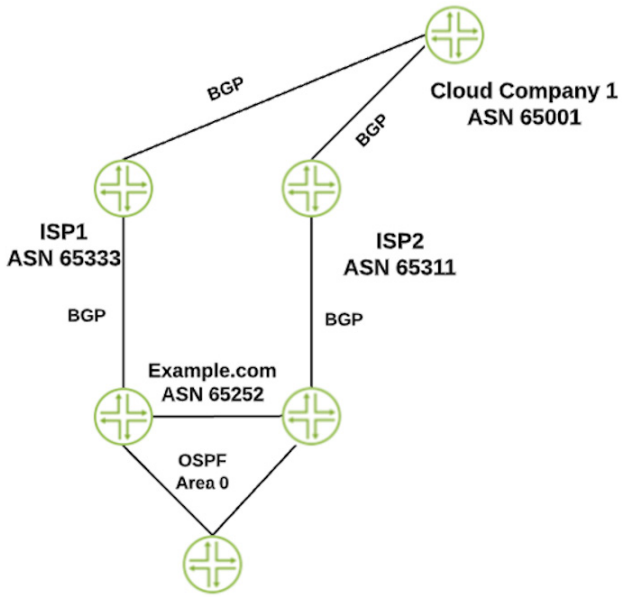


Figure 5.14 *BGP as EGP*

And dynamic routing protocols also allow for the ability to administratively override a path if needed through cost or routing policy, which is dealt with in Chapter 6.

OSPF

Open Shortest Path First (OSPF) is a link-state protocol commonly used on enterprise networks. Each router will flood *Link State Advertisements* (LSAs) that will describe that device's view of the routed network. It floods these LSAs through an administrative segment called an *Area*. OSPF runs the Dijkstra algorithm within each area to calculate the best path and you can use multiple areas for scalability as well as to create a hierarchy such as shown in Figure 5.15.

Figure 5.15 *OSPF as an IGP*

You can see in Figure 5.16 that OSPF floods these LSAs to each router in the area. This creates a complete LSA database for the entire area. In Figure 5.16 the vSRX1 will flood its LSAs to each of its connected neighbors. They will flood this information to any other connected OSPF routers as well. This will continue until the entire area has a complete copy of all LSAs.

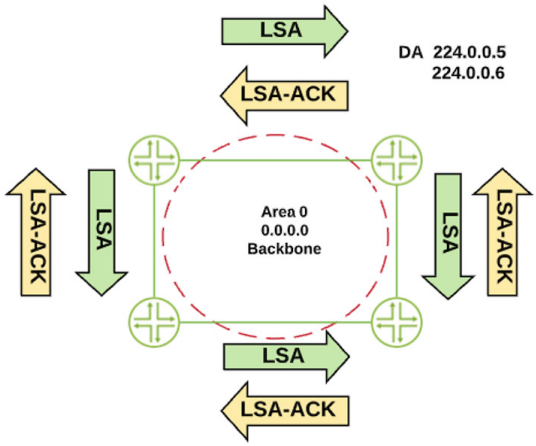


Figure 5.16 LSA Flooding

As you configure OSPF on Junos you have to be aware of some slight optic differences. For one, all of the configuration happens in a single place, under the protocol OSPF hierarchy. No need to hunt all over your config for different elements here. The only time that changes is if you are using a routing instance. Then OSPF for that instance is contained under its own `routing-instance instance-name protocol ospf` knob.

The logical interface is important with any configuration under Junos. So please pay attention to the unit number. You will see that if you do not add the logical unit number to parts of the configuration, Juniper will make a decision for you. In this case it will automatically add the `.0` to the end of the physical interface. The loopback interface is not automatically added to the OSPF network. This might be important if you have configured a manual router ID for use as identification and sourcing of your dynamic routing protocols. You may want to use your loopback for this purpose.

Loopbacks are always up and most likely one of the last places you may change an IP address. In our example let's configure the loopback interface as a passive interface. This adds the interface as an internal route and it also prevents neighbor relationships from establishing, which doesn't make a lot of sense on a loopback but may make sense on an edge device. The area can be configured with its short cut

value such as in the case of Area 0. Though be aware that it will represent itself as a 32-bit area of 0.0.0.0. This doesn't change the way it works, but it may change your ability to decipher some crazier area numbers as you move above 255.

So let's configure a single area 0 with two interfaces and a loopback. The interfaces do not have the `passive` option because we would like to form neighbor relationships across them. Here, we also add the loopback as a passive interface, allowing this network to be advertised as an internal OSPF route with a preference of 10:

```
[edit protocols ospf]
lab@vSRX1# set area 0 interface ge-0/0/0.0
lab@vSRX1# set area 0 interface ge-0/0/1.0
lab@vSRX1# set area 0 interface lo0.0 passive
```

In the example below we see as we review the config, we see how it appears after we make our additions. We see that even though we defined the area as a 0 it automatically changed it to 0.0.0.0. We see the interfaces being added and the passive interface as well.

```
[edit protocols ospf]
lab@vSRX1#show
area 0.0.0.0    >>> Junos will convert this to a 32 bit value.  You can still use area 0.
    interface ge-0/0/0.0   >>> the unit number here is important.
    interface ge-0/0/1.0
    interface lo0.0
passive  >>>Loopback addresses do not automatically advertise under Junos.  You need to configure
passive.
  }
```

Now, after we commit on the vSRX1 and also one of its neighbors, the OSPF adjacencies come up and we should start to see OSPF routes. All of our OSPF internal routes come in as preference 10. If we had a redistribution into OSPF from another routing protocol, we would see that as an external OSPF route with a preference of 150.

Now let's verify that the OSPF neighbors are up:

```
lab@vSRX1>show ospf neighbor
Address          Interface              State     ID               Pri  Dead
172.31.254.2     ge-0/0/0.0             Full      10.1.1.1         128    30
172.31.253.2     ge-0/0/1.0             Full      10.1.1.1         128    39
```

Another thing to be aware of is the metric. By default, each link it crosses with accumulating a cost of one which will be referenced as a metric. You can artificially change this by changing the metric or cost for an interface. Be careful with it, however, as we run Dijkstra across the entire area and these changes can influence the path through your device. These costs or metrics are applied in one direction as well. If you influence a path by changing a cost on a link you may introduce asymmetrical packet flow unintentionally.

Let's look at the route for 10.2.2.0/30. There are two OSPF routes. By default,

only one of them will become the active route and will be installed in the forwarding table. If you would like to use both next hops you would need to configure a load balancing policy for the forwarding table:

```
lab@vSRX1> show route protocol ospf

inet.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.2.2.0/30        *[OSPF/10] 00:01:21, metric 2
                       to 172.31.254.2 via ge-0/0/0.0
                    >  to 172.31.253.2 via ge-0/0/1.0
224.0.0.5/32       *[OSPF/10] 00:02:33, metric 1
                       MultiRecv
```

Remember when we started talking about OSPF we mentioned that it uses LSAs to create a database? You can validate this by checking the OSPF database:

```
lab@vSRX1> show ospf database

    OSPF database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router   10.1.1.1         10.1.1.1         0x80000004   140  0x22 0x8832  60
Router  *172.31.253.1     172.31.253.1     0x80000004   140  0x22 0x3d20  48
Network *172.31.253.1     172.31.253.1     0x80000001   140  0x22 0x9b34  32
Network *172.31.254.1     172.31.253.1     0x80000001   145  0x22 0x903e  32
```

Here you can see four LSAs across the two connected routers. The asterisk references that this LSA was generated by the router you are running this operational mode command on.

When you dig a bit deeper you can see how these LSAs can build a topology within an area. Next is the output for the vSRX2s router LSA. This LSA states that there are two transit networks and one stub. The stub if you remember is the passive interface. The two transit interfaces reference the fact that there is another OSPF router on the other side of that link:

```
lab@vSRX1> show ospf database advertising-router 10.1.1.1 detail

    OSPF database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router   10.1.1.1         10.1.1.1         0x80000004   326  0x22 0x8832  60
  bits 0x0, link count 3
  id 172.31.254.1, data 172.31.254.2, Type Transit (2)
    Topology count: 0, Default metric: 1
  id 172.31.253.1, data 172.31.253.2, Type Transit (2)
    Topology count: 0, Default metric: 1
  id 10.2.2.0, data 255.255.255.252, Type Stub (3)
    Topology count: 0, Default metric: 1
  Topology default (ID 0)
    Type: Transit, Node ID: 172.31.253.1
      Metric: 1, Bidirectional
    Type: Transit, Node ID: 172.31.254.1
      Metric: 1, Bidirectional
```

Chapter 6

Routing Policies

By Yasmin Lara

Understanding routing policies in Junos is a critical skill for you to be able to successfully configure and troubleshoot an extensive list of features and protocols running on your Juniper devices.

What Is Routing Policy?

Routing policies are primarily known as a way to control routing information in and out of the routing table, after all they are called routing policies for a reason. However, you will be surprised to find that routing policies are used to control a variety of features and operations in your Juniper device that range from load balancing, to multicast, and even to CoS.

This chapter focuses on routing policies to control routing information, but Table 6.1 gives you a good idea of how extensive the list of features that require the use of policies is.

Table 6.1 Routing Policy Applications

Feature	What Policies Are Used For.
Load balancing	To control which routes are installed in the forwarding table. To configure the router to install all available next hops for a particular route(s).

Route redistribution between protocols	To define which routes are redistributed from one protocol to another. To define the attributed of the routes redistributed. Example: The type of OSPF external route after the route is redistributed from BGP into OSPF.
Route sharing between routing instances	To control which routes are installed in each routing table when configuring rib-groups or using instance-import to share routes between different routing tables.
BGP attributes	To change/fix the next-hop for a particular route. To implement Autonomous System prepending (AS-prepend), add/delete communities from the routes, change the origin, or change the MED. To change a route attribute based on another attribute. Example: change the local preference, based on a community value. To configure Route aggregation (redistribution of aggregate routes into BGP).
L3 VPN/MPLS	To control which prefixes get advertised by LDP. To control customer routes redistribution from and into BGP. To map traffic to a particular LSP (label-switched path) by installing routes using the LSP as next hop into the forwarding table.
ISIS	To leak routes from ISIS level 2 to Level 1; filter routes from Level 2 to Level 1. To control which prefixes get advertised by ISIS. To configure Route aggregation.
OSPF	To control the creation and propagation of LSAs type 3 in an ABR. To control which external routes are installed in the routing table from the SPF calculations (from LSAs type 5).
Multicast	To control the propagation of IGMP, PIM, and MSDP messages. To control the routing information used for RPF verifications.
Class of Service based forwarding	To map destination prefixes to different next-hops or LSPs based on traffic type (forwarding class).

You can see how valuable routing policies are, so let's get into the details of how they work. Again, we will be focusing on their use within the context of routing protocols.

Routing Policies to Control Routing Information Flow

The first two things you need to know about routing policies and their use within the context of routing protocols are:

1. Routing policies control the flow of routes to and from the routing table. There are two important concepts here: *It is ALWAYS from the point of the routing table, and affects ROUTES!*

2. They can be used to accept, reject, or modify attributes *for routes*, received from neighbors, or sent to neighbors, depending on how the policies are applied. Again, they affect ROUTES!

Import Policies Versus Export Policies

To control the flow in and out of the routing table you apply routing policies as either IMPORT policies or EXPORT policies. Figure 6.1 illustrates how this works:

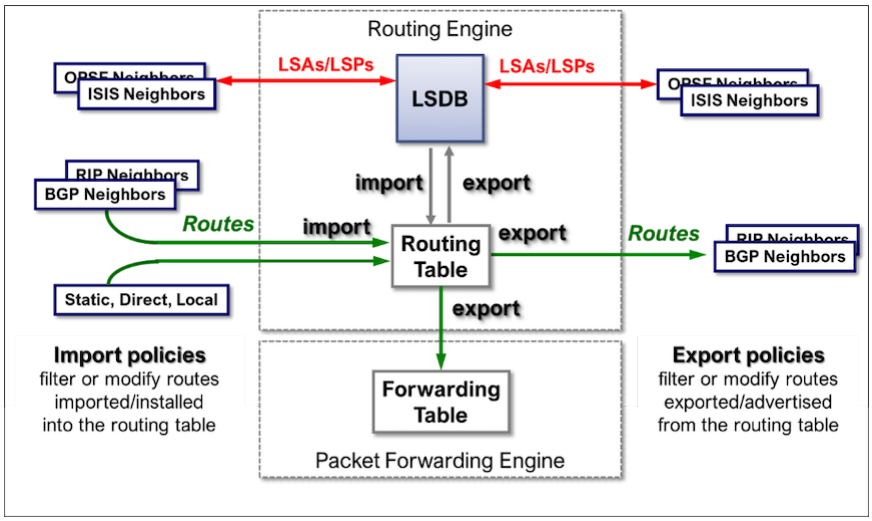


Figure 6.1 *Apply Routing Policies*

When you look at Figure 6.1 you should notice that the router can be exchanging routing information with neighbors in different ways, depending on the routing protocol. If the router is running BGP, or perhaps even RIP, the updates actually contain routes or prefixes. If the router is running a link-state protocol (LSP) such as OSPF, rather than advertising routes, the routers exchange LSAs/LSPs which describe the links between routers. In Chapter 5 we talked about basic OSPF, and described how information is advertised using LSAs which are placed into the link-state database (LSDB).

We will *not* cover these details as it goes beyond the scope of this book, but it is important that you keep in mind that applying routing policies to Link State Routing Protocols works a little differently.

Our interest right now is in the flow of routes. Let's first look at the import direction.

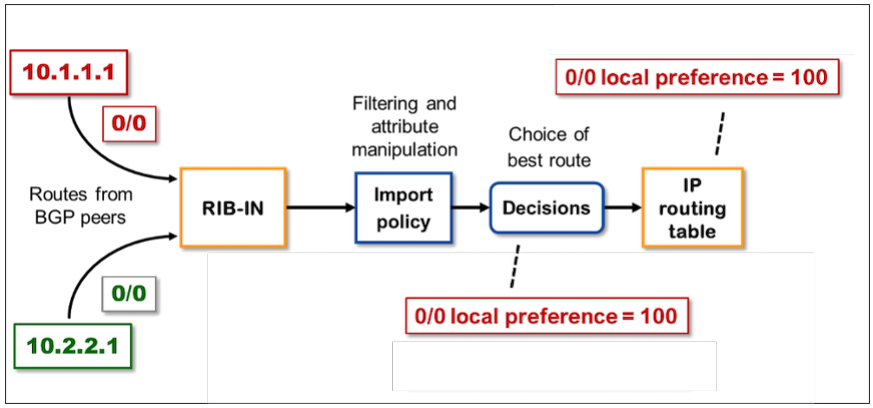


Figure 6.2 *Flow of Routes Import*

You can see how we might be receiving routing information from multiple neighbors, from multiple routing protocols, and from multiple routing information sources (including interface routes, static routes, and so on) all at the same time. The router needs to decide whether these routes should be installed in the routing table. Each protocol or source of information has its own rules. Remember, for example, that static routes require that the next hop be reachable. BGP will check that the route has not been looped by checking the list of autonomous systems the route has gone through. An import policy can be applied to the routes before the router makes these decisions, and before the routes are installed in the routing table. This policy can reject a route that otherwise would have been accepted, which results in the route *not* being installed in the routing table. The policy can also change the route's attributes before it is installed in the routing table, which can affect which route is selected as the active route. For example, consider Figure 6.3.

The router is receiving a default route from two different BGP neighbors: 10.1.1.1 and 10.2.2.1. The two routes are accepted by default but an import policy changes the local preference for the route received from 10.2.2.1 to 200, making this route better for the BGP selection process, and therefore making this route the active route in the routing table (indicated with the *).

An import policy could also reject the default route received from 10.1.1.1 and only accept the one from 10.2.2.1, without changing any attribute. The route from 10.2.2.1 would become the active route and the one from 10.1.1.1 would become a hidden and unusable route (Figure 6.4).

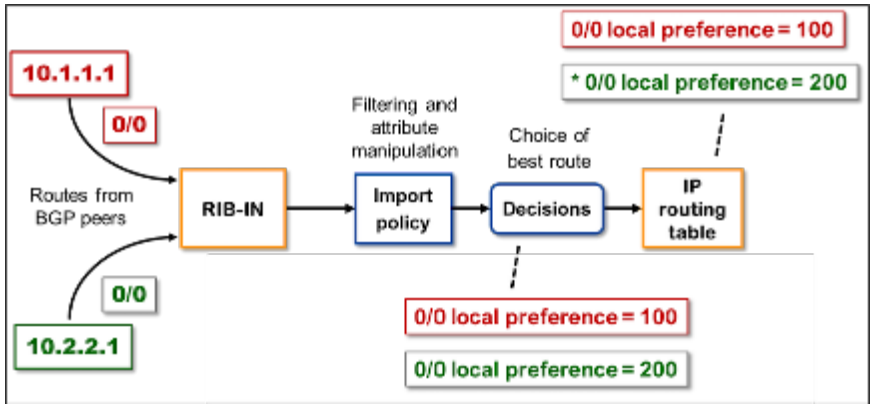


Figure 6.3 *Change the Route's Attributes*

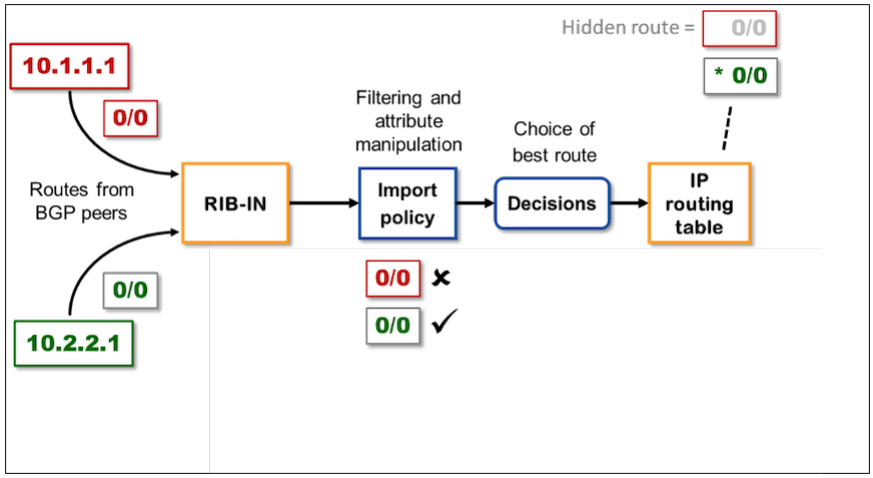


Figure 6.4 *Import Policy Rejects Default Route*

Let's now focus on the export direction.

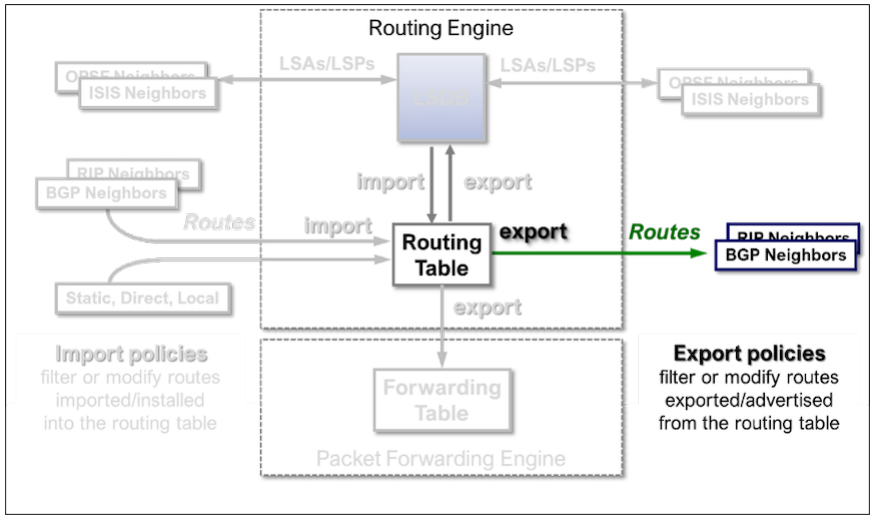


Figure 6.5 *Export Direction*

What we are looking at in Figure 6.5 is how the router is taking *routes already in the routing table* and sending them to neighbors. The question here is: which routes should be sent to which neighbor? And in the process, should the attributes of the routes be kept the same or should they be modified to influence the decisions made by those neighbors?

The export policy will determine that, possibly also allowing the redistribution or translation of routes from one routing protocol to another, the advertisement of routes that would not be advertised by default or preventing the advertisement of routes that would otherwise be advertised.

For example, let's say we have a default route that we learned from BGP and a route to 172.16.1/24 that we learned from OSPF. Both are active in the routing table. We might want to advertise the default route only to our neighbor 10.4.4.1, and the route to 172.16.1/24 only to our neighbor 10.3.3.1, as shown in Figure 6.6.

We would need to apply policies that redistribute the OSPF route into BGP and only send it to 10.3.3.1, and also prevent the advertisement of the default route to 10.3.3.1 while allowing its advertisement to 10.4.4.1.

Another important concept is this: The effect of your policy action is visible only after the route is imported into or exported from the routing table.

However, you can use the `show route receive-protocol bgp <neighbor>` command, to view the BGP routes that were received from a neighbor, *before* the routing policy is applied. You can also use the `show route advertising-protocol bgp <neighbor>` command to view the BGP routes that will be advertised to the neighbor, *after* the export policy is applied.

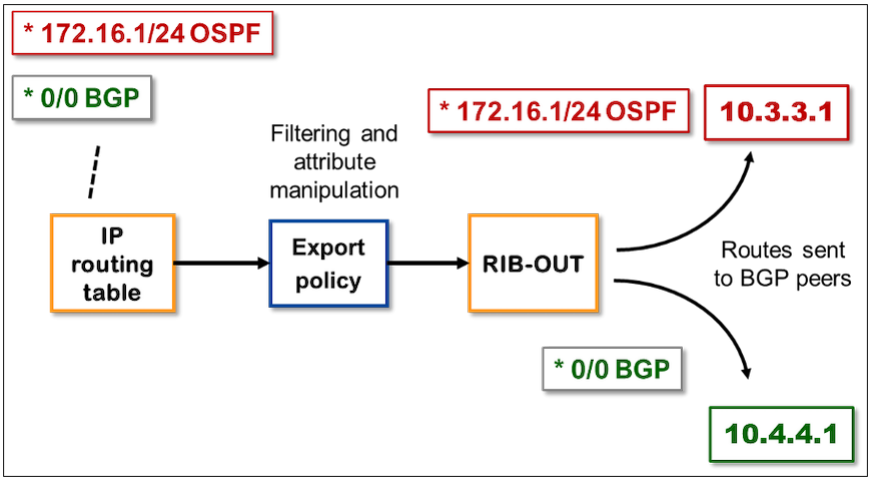


Figure 6.6 *Routes Sent to BGP Peers*

Export Policy Applied to the Forwarding Table (Load Balancing in Junos)

Referring back to the previous figures, you probably also noticed that there is an export policy applied between the routing table and the forwarding table. Let's now talk about that, as is shown in Figure 6.7.

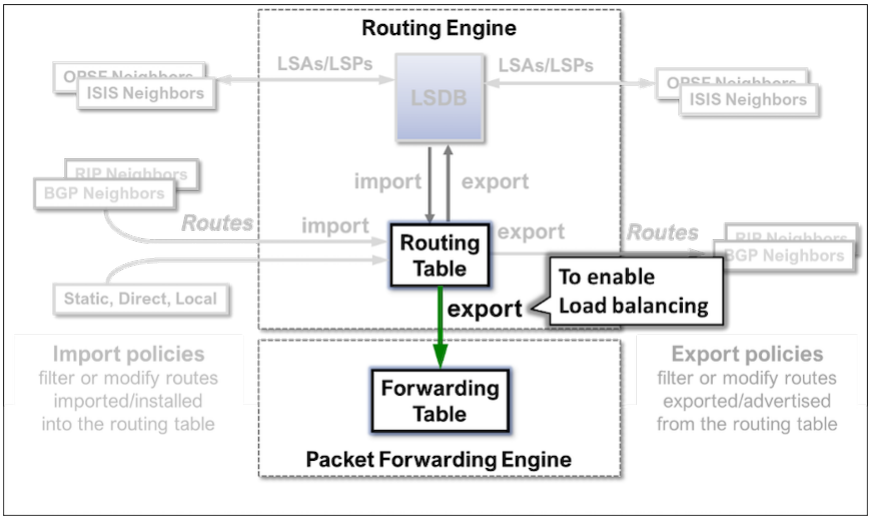


Figure 6.7 *Export Policy Between Tables*

Remember from our earlier discussion about the router architecture that the routing table is built by the RE, and contains routes from all the different sources of information enabled on the router (BGP, static, direct, and so on). The RE selects the best route for each destination based on preference values, metric, and other attributes, and these selected routes become active and are copied into the forwarding table, which resides in the PFE.

By default, only one next hop is selected for each of these active routes, so if, for example, you have a route with two or more next hops (equal cost), only one of those next hops will be installed in the forwarding table. Thus, even when there are multiple next hops equally good and known by the router to reach a given destination network, only one of them is used to forward traffic. In other words, load balancing is not implemented by default in Junos.

In the example in Figure 6.8 you can see that even though there are two next hops to reach 10.1.1/24 (R1 and R2), the forwarding table only has R1.

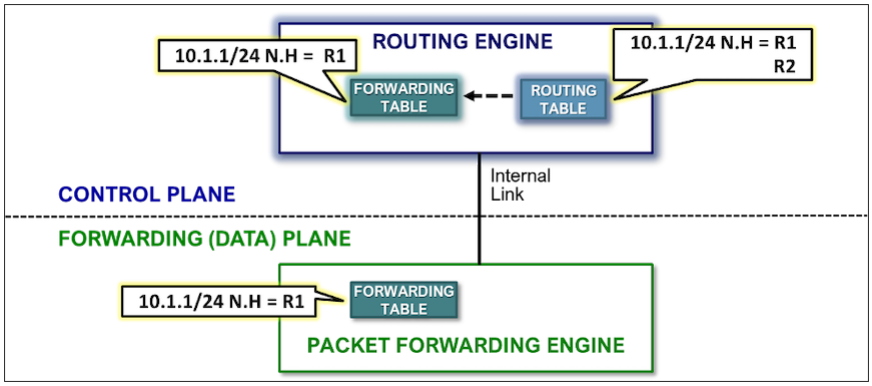


Figure 6.8 Default Behavior

You can change these behaviors and enable load balancing by applying a policy between the two tables as shown in Figure 6.9.

This policy could be as simple as:

```
set policy-options policy-statement load-balance term 1 then load-balance per-packet
set routing-options forwarding-table export load-balance
```

Don't worry about the syntax of the policy for now, we will talk about that in a lot of detail later.

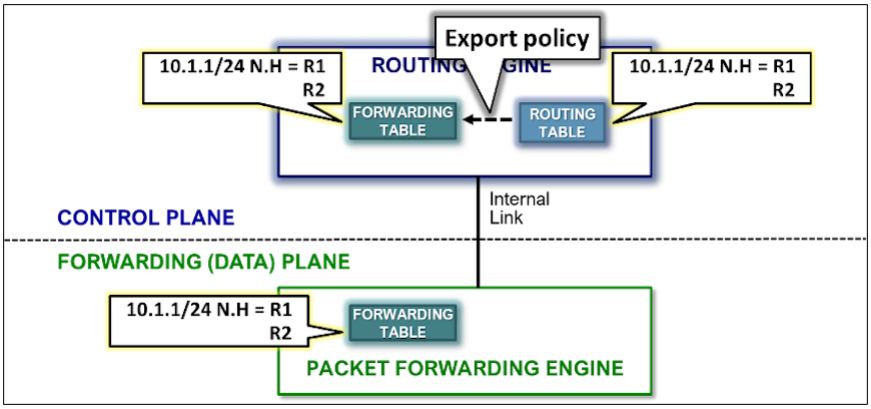


Figure 6.9 *Enabling Load Balancing*

What's important here is that you realize that *this policy is also applied as an EXPORT policy*. Remember one of those two important concepts we described at the beginning: *it is ALWAYS from the point of view of the routing table.* We are taking routing information *from the routing table* and placing it in the forwarding table, so we apply the policy to the forwarding table but as an export policy.

NOTE All the sample scenarios we have seen so far describe user-defined import policies (policies that you wrote), but you should know that there are also default import and export policies which will be there. When you write your own you are only overriding the default behavior one way or another. This is covered in a lot of detail later.

How to Tell If a Policy Should Be an Export Policy or an Import Policy

Before we move on, let's look at the import versus export idea one more time. If at any time you are not sure whether a policy needs to be applied as an EXPORT or as IMPORT policy remember: It's always from the point of the routing table as shown in the sequence of Figures 6.10.

Are we ready to learn how to write policies?

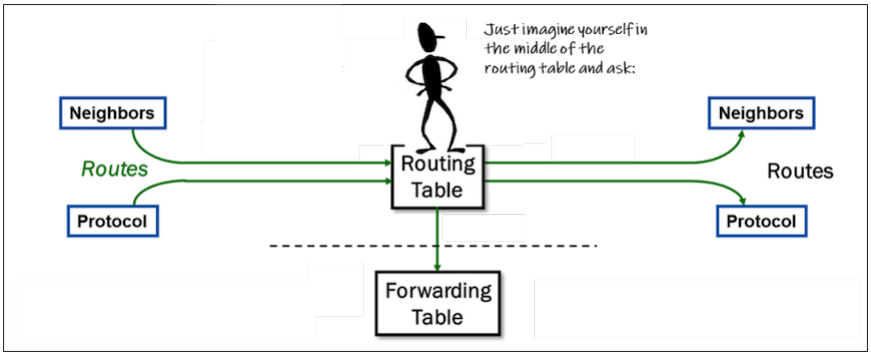


Figure 6.10 *Point of Reference is the Routing Table*

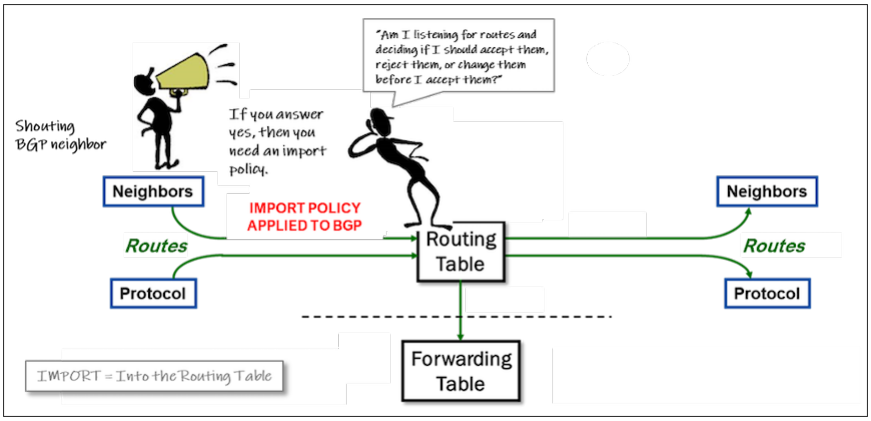


Figure 6.10 *Import=Into the Routing Table*

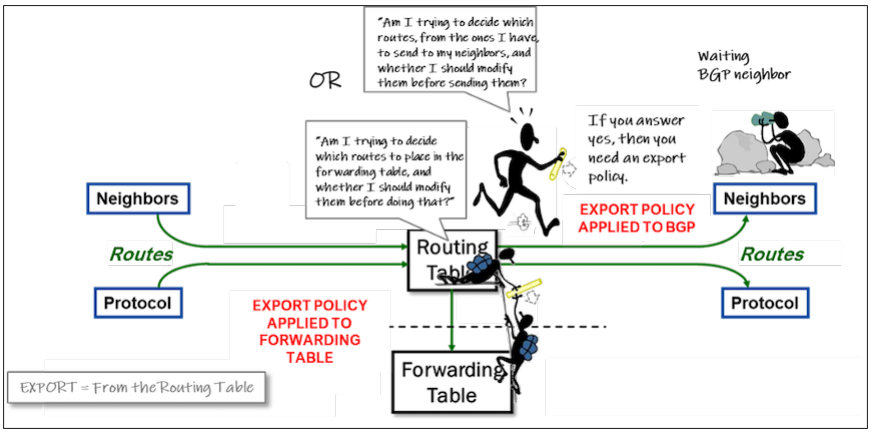


Figure 6.10 *Export=From the Routing Table*

Policy Components

Terms

A routing policy is a sequence of terms(the building blocks) that contain one or more matching criteria, and one or more actions to be taken when such criteria is met. Let's look at Figure 6.11.

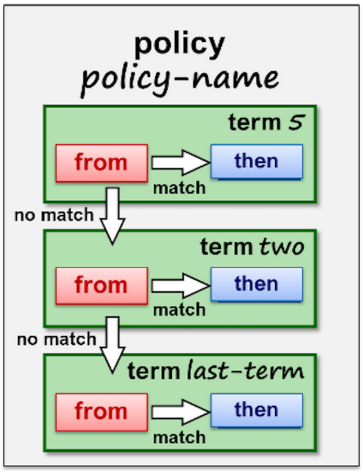


Figure 6.11 *Policy's Building Blocks*

To create your policy, go under [policy-options] and give the policy a name (whatever you want), which you will then use to reference the policy when you are applying it under a given routing protocol.

For example, you can create a policy named `bgp-to-ospf` and then apply it to `ospf`:

```
[edit policy-options]
user@router# set policy-statement bgp-to-ospf ….

[edit protocols ospf]
user@router# set export bgp-to-ospf
```

You can have *zero or more terms* within your policy. And yes, *zero* is correct here because you have the option of omitting the term and defining your matching criteria and actions directly under the policy.

Best practice does not recommend this. It is always better to use terms from the beginning, even if you are writing a simple policy. You will save yourself some headaches later if you need to modify the policy for any reason.

You create the terms by giving them unique names within the policy, which again

are completely up to you. The name can be anything including letters, numbers, and special characters:

```
[edit policy-options policy-statement TEST-POLICY]
user@R1# show | display set relative
set term 123 then accept
set term FIRST then reject
set term LAB123 then accept
set term term_#2 then reject
```

Keep in mind that using numbers does not provide any sequencing information. If, for example, we create the following policy and add terms 5, 2, and 6, in that order:

```
[edit policy-options policy-statement TEST-POLICY2]
user@R1# set term 5 then accept

[edit policy-options policy-statement TEST-POLICY2]
user@R1# set term 2 then accept

[edit policy-options policy-statement TEST-POLICY2]
user@R1# set term 6 then accept
```

The result is that the terms are listed in the same order we entered them:

```
[edit policy-options policy-statement TEST-POLICY2]
user@R1# show | display set relative
set term 5 then accept
set term 2 then accept
set term 6 then accept
```

Any term that we add, no matter what we call it, will always be added to the bottom.

If we configure term 1 now:

```
[edit policy-options policy-statement TEST-POLICY2]
user@R1# set term 1 then accept

[edit policy-options policy-statement TEST-POLICY2]
user@R1# show | display set relative
set term 5 then accept
set term 2 then accept
set term 6 then accept
set term 1 then accept
```

You can see it was added at the end of the policy.

You can reorder terms with one of the CLI tricks learned in Chapter 2:

```
[edit policy-options policy-statement TEST-POLICY2]
user@R1# insert term 1 before term 2

[edit policy-options policy-statement TEST-POLICY2]
user@R1# show | display set relative
```

```
set term 5 then accept
set term 1 then accept
set term 2 then accept
set term 6 then accept

[edit policy-options policy-statement TEST-POLICY2]
user@R1# insert term 5 after term 2

[edit policy-options policy-statement TEST-POLICY2]
user@R1# show | display set relative
set term 1 then accept
set term 2 then accept
set term 5 then accept
set term 6 then accept
```

But why is this important?

Routes that are being evaluated against a policy, are *evaluated term by term sequentially, and in strictly top to bottom fashion*, until a match with a terminating action (which we'll cover later) is found or the end of the policy is reached. If the order of the term is not correct, the policy might reject routes that should be accepted, or accept routes that should be rejected (See Figure 6.12).

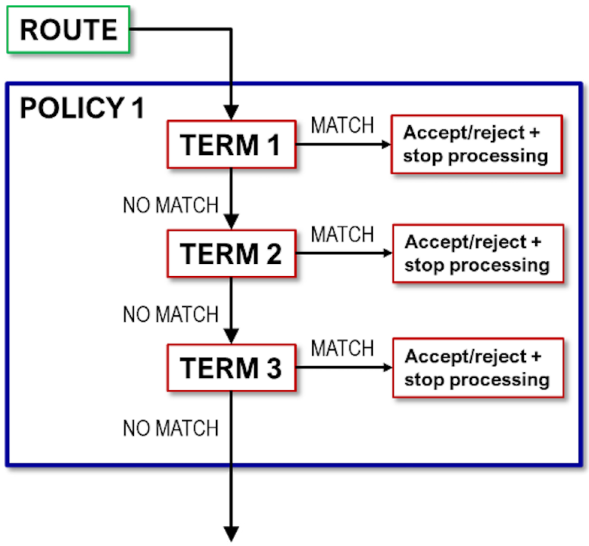


Figure 6.12 Policy Processing Sequence

Imagine that you need to create a policy that allows subnet 172.16.1.0/24 and rejects any other 172.16/16 route, but you configure the policy like this:

```
[edit policy-options policy-statement ALLOW-SUBNET]
user@R1# show | display set relative
```

```
set term AGGREGATE from route-filter 172.16.0.0/16 exact
set term AGGREGATE then reject
set term SUBNET from route-filter 172.16.1.0/24 exact
set term SUBNET then accept
```

NOTE We'll talk about this syntax in detail soon. For now, just pay attention to the order of the term and the prefixes they are matching on.

The result of applying this policy: when the router evaluates a route for 172.16.1/24 for example, it will match the first term (AGGREGATE), the route will be rejected, and that would be it for this route. No further processing. Any other subnet of 172.16/16 will have the same fate. Mission failed, though easy to fix! We reorder the terms using:

```
[edit policy-options policy-statement ALLOW-SUBNET]
user@R1# insert term SUBNET before term AGGREGATE

[edit policy-options policy-statement ALLOW-SUBNET]
user@R1# show | display set relative
set term SUBNET from route-filter 172.16.1.0/24 exact
set term SUBNET then accept
set term AGGREGATE from route-filter 172.16.0.0/16 exact
set term AGGREGATE then reject
```

And now, when the router evaluates a route for 172.16.1/24 it will match the first term (`SUBNET`), and the route will be accepted. Any other subnet of 172.16/16 will not match the first term, but will match the second term, and will be rejected, as we wanted. Mission accomplished.

Now within each term, you can define the matching criteria with one or more `from` statements, and the action(s) with one or more `then` statements.

Let's dive into how we can define this matching criteria and which actions we can assign.

Matching Criteria – The From Statement

Since we are talking about using policies to control routing information, we are looking at routes, that is the destination network or prefix, and the attributes that describe how to reach that destination.

Common Match Criteria

Since we are looking at routes, the matching criteria can include:

- Prefix:
- Using a route-filter, a prefix-list, a prefix-list-filter or a route-filter-list.

Protocol or route information source such as:

- OSPF
- Static
- BGP
- aggregate

Routing protocol specific attributes such as:

- OSPF area ID
- AS path
- MED/metric
- BGP community
- ISIS level
- Route type: internal or external
- Interface
- Next hop

Within a term you can define any number of `from` statements. If you omit the `from` statement completely, all routes match the term. In case of multiple `from` statements all the matching criteria must match for the route to match the term, but how does the router determine what matching all the matching criteria means? This is in the next section.

NOTE The `from` statement is optional.

Multiple From Statements – An AND or an OR?

When you define a term with multiple `from` statements there are two possible combinations:

1. You can have multiple `from` statements of the same type, or matching the same attribute. For example:

```
[edit policy-options policy-statement TEST]
user@R1# show | display set relative
set term 1 from protocol bgp
set term 1 from protocol static
```

In this example, if the route is BGP OR static it matches the term.

2. You can have multiple `from` statements of different types or matching different attributes. For example:

```
[edit policy-options policy-statement TEST]
user@R1# show | display set relative
set term 2 from protocol direct
set term 2 from interface ge-0/0/0.0
```

In this example, if the route is direct AND the outbound interface is ge-0/0/0.0, it matches the term.

Now, Figure 6.13 shows both combinations together.

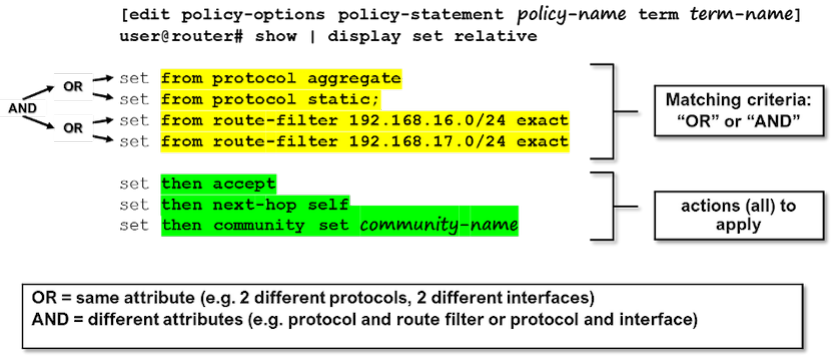


Figure 6.13 Combining Multiple From Statements

In Figure 6.13, a route must be either an aggregate route, OR a static route AND the prefix must be either 192.168.16/24 OR 192.168.17/24 to match the term.

Now, matching the protocol, or the interface, or the metric are quite simple criteria, but what about matching prefixes? You have seen some examples already but matching prefixes can be far more complicated.

There are four different options to match prefixes within the context of routing policies:

- Prefix-list
- Prefix-list-filter
- Route-filters, and
- Route-filter-lists

In general, these options match routes by specifying the number of bits that need to be the same and also the length of the subnet mask. What does that mean?

Let's say you are receiving the following routes from your neighbors:

```
172.16.1.0/24      A subnet of 172.16.0.0/16
172.16.1.0/25      A subnet of 172.16.1.0/24
172.16.1.128/25  A subnet of 172.16.1.0/24
```

```
172.16.1.0/26      A subnet of 172.16.1.0/24
172.16.1.128/26  A subnet of 172.16.1.0/24
172.16.1.140/26  A subnet of 172.16.1.0/24
172.16.2/24        A subnet of 172.16/16
172.16.2.128/26  A subnet of 172.16.2.0/24
```

Now, let's say that you need a policy that only accepts routes for the subnets of *172.16.1.0/24 that have a subnet mask of /25 (172.16.1.0/25 and 172.16.1.128/25*), while rejecting any other subnet.

You can see that both routes begin with 172.16.1 (the first 24 bits match 10101100.0010000.00000001, yes, 172.16.1 in binary), and of course they have a subnet mask = /25, though bit 25 is *not* the same.

I could match each of these two routes individually, that would not be too hard, but imagine a situation where you want to do something similar but there are many more routes that you want to match on. It becomes impractical to add each prefix individually.

Thus, when you want to match multiple routes, you can use something called route filters, which has two components:

```
<prefix>            Defines the number of bits that need to match
<matching-type>     Defines the length of the subnet mask that needs to match
```

NOTE You can think of Matching-type as subnet-mask-matching-type.

This `subnet-mask-matching-type` can be `exact`, `longer`, `orlonger`, `upto`, `through`, `prefix-length-range`, and `address-mask`. Let's look at these different options with the following examples:

Table 6.2 Matching Type Examples

Matching Type	Example	
Exact	192.168.0.0/16 exact	Routes matching 192.168.0.0 in the first 16 bits AND with a subnet mask of exactly 16 bits. => Only routes for exactly 192.16.0.0/16 match.
Longer	192.168.0.0/16 longer	Routes matching 192.168.0.0 in the first 16 bits AND with a subnet mask LONGER than 16 bits. => Any subnet of 192.16.0.0/16 (e.g 192.168.0.0/17, 192.168.1.0/17, 192.168.0.0/18, 192.168.100/24) matches. => 192.168.0.0/16 does NOT match.

Orlonger	192.168.0.0/16 orlonger	A route matching 192.168.0.0 in the first 16 bits AND with a subnet mask of 16 bits ORLONGER. => Any subnet of 192.16.0.0/16 (e.g 192.168.0.0/17, 192.168.1.0/17, 192.168.0.0/18, 192.168.100/24) matches. => 192.168.0.0/16 ALSO matches
Upto	192.168.0.0/16 upto /24	A route matching 192.168.0.0 in the first 16 bits AND with a subnet mask between 16 and 24. => Any subnet of 192.16.0.0/16 with a subnet mask between 16 and 24 (e.g 192.168.0.0/16, 192.168.1.0/17, 192.168.0.0/18, 192.168.100/24) matches. => 192.168.0.0/16 matches. => 192.168.1.128/26 does NOT match
Prefix-Length-Range	192.168.0.0/16 prefix-length-range /20-/24	A route matching 192.168.0.0 in the first 16 bits AND with a subnet mask between 20 and 24 bits. => Any subnet of 192.16.0.0/16 with a subnet mask between 20 and 24 (e.g 192.168.0.0/20, 192.168.64.0/22, 192.168.128.0/23, 192.168.10/24) matches. => 192.168.0.0/16 does NOT match. => 192.168.1.128/26 does NOT match. => 192.168.1.0/17 does NOT match.
Through	192.168.1.0/24 through 192.168.1.192/27	Only prefixes 192.168.1/25, 192.168.1.128/25, 172.16.1.192/26, and 192.168.1.192/27. This option is not covered in the current version of Junos, and its use is not very common, but you will see it when you look at all the configuration options for route-filters, using ? on your router, so we might as well just make you aware of it.
Address Mask	172.16.1.0/24 address-mask 255.0.255.0	This option is not covered in the current version of Junos, and can become very complex. Just know that you will see this option if you use the ? and that you can configure things that look as crazy as the example, which basically means the route needs to match 172.0.1. Enough said!

When you are evaluating whether a route matches a filter or not, you need to follow a two-step process. For example, if you have a filter that says: `192.168.128.0/17 exact`, and you are checking if a route for 192.168.192.0/24 matches or not, follow these steps:

Step 1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask)

If you configure filter 192.168.128.0/17 the first 17 bits of the route must match the first 17 bits on 192.168.128.0. (see Figure 6.14 Step 1).

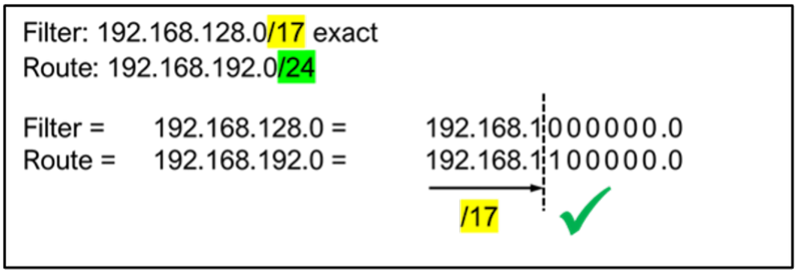


Figure 6.14 Step 1

Step 2: Compare the number of bits in route's subnet mask with the number of bits in the filter's subnet mask according to the subnet mask match type configured in the filter.

It is pretty obvious that the subnet mask of the route (/24) in our previous example is NOT the same as the subnet mask in the filter (/17). Because the route filter is configured with a match type = exact, the *route does not match* the filter as it does not meet the criteria on the two steps. See Figure 16.4 Step 2.

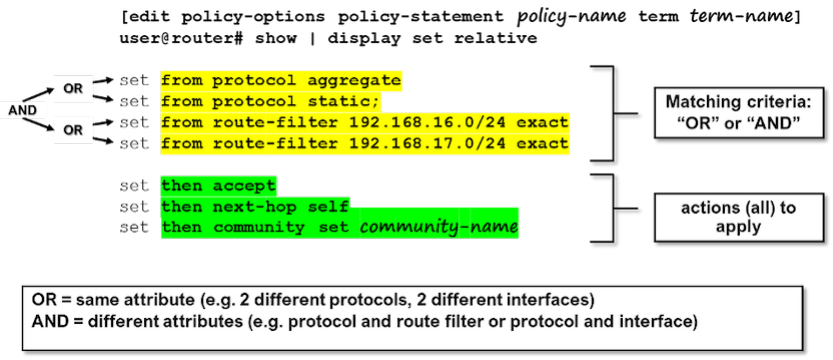


Figure 6.14 Step 2

If we changed the filter match-type to `longer`, or to `orlonger`, the result would be different. Step 1 would be the same but now in Step 2, because the subnet mask in the route (/24) is longer than the subnet mask of the filter (/17), the route also meets the criteria. Thus, the route matches the filter as shown in Figure 6.14 Step 2 with Longer Instead of Exact.

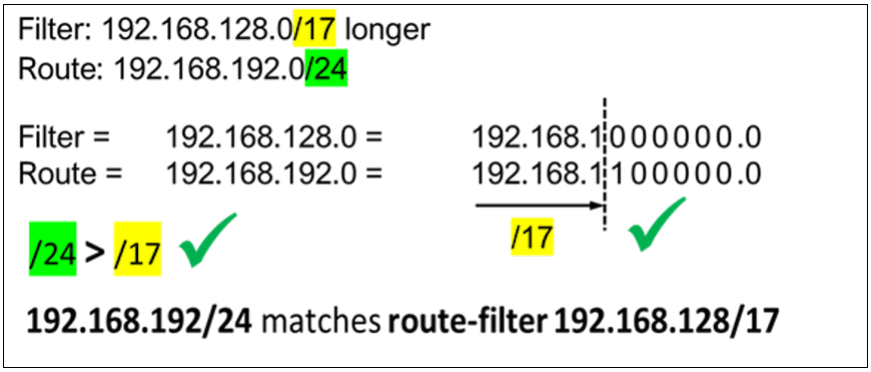


Figure 6.14 Step 2 with Longer Instead of Exact

But, how does the router implement this?

The router uses a data structure called a *radix tree* to represent every possible combination of 32 bits and subnet mask values as shown in Figure 6.15.

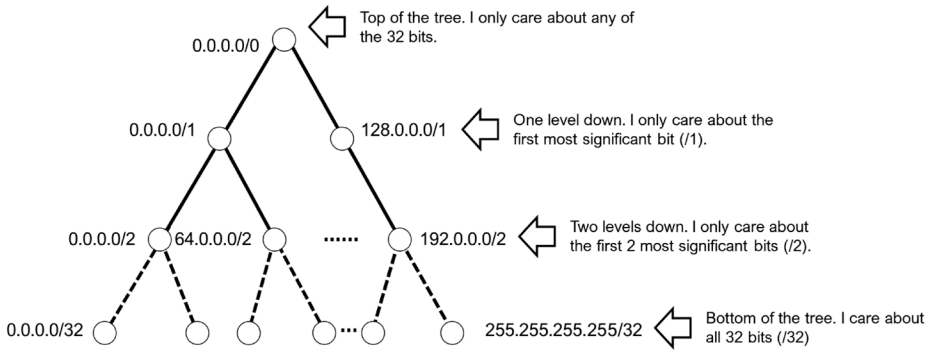


Figure 6.15 Radix Tree

When you write a filter, the router finds the prefix in the tree, and any route under that point in the tree, and within the subnet match type criteria, will match the filter.

For example, of the filter is `172.16.1/24 exact`, a route has to match that exact point in the radix tree as shown in Figure 6.16.

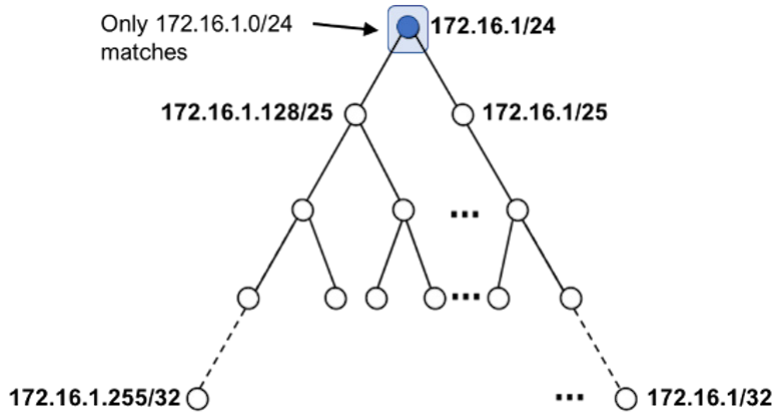


Figure 6.16 *172.16.1.0/24 Longer Exact*

If the -filter is `172.16.1/24 orlonger`, a route has to match the exact point in the radix tree or any point *under* it (see Figure 6.17).

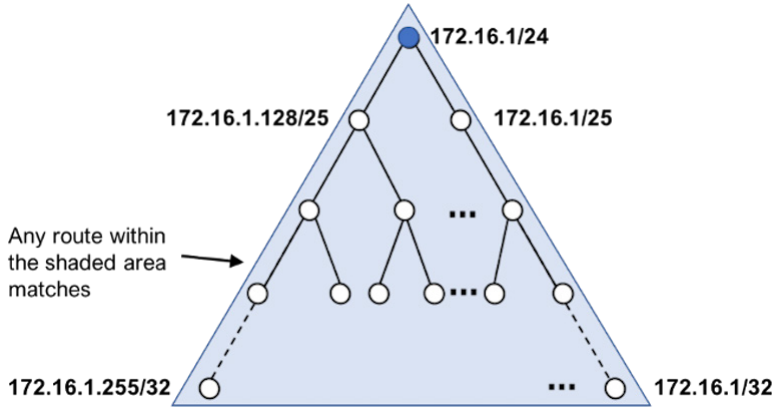


Figure 6.17 *172.16.1.0/24 or longer*

The shaded area represents the area where the route needs to be located to match the filter. In other words, a route matches if it matches any of the points within the shaded area. If the filter is `172.16.1/24 longer`, a route has to match any point under 172.16.1/24, in the radix tree. A route that matches the exact point, does not match the filter. See Figure 6.18.

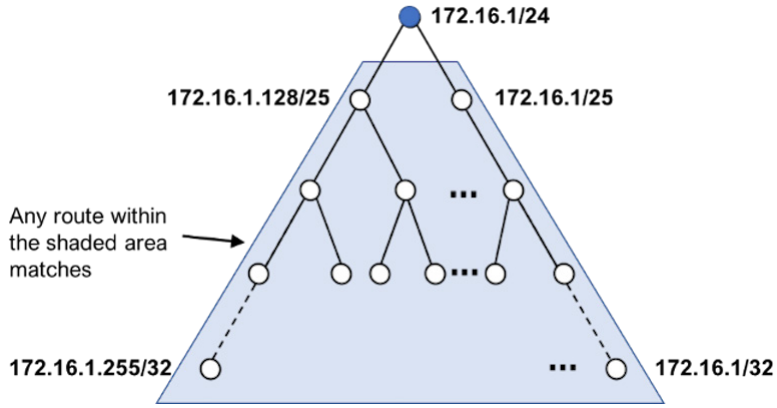


Figure 6.18 *172.16.1.0/24 longer*

If the filter is `172.16.1/24 upto /27`, a route has to match the exact point in the radix tree or any point under it, but up to prefix length = 27 (/27) as in Figure 6.19.

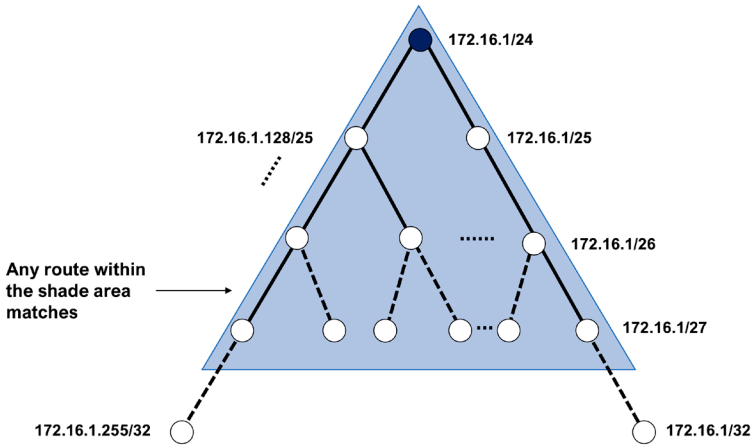


Figure 6.19 *172.16.1.0/24 upto /27*

If the -filter is `172.16.1/24 prefix-length-range /26-/27`, a route has to match any point under 172.16.1/24 in the radix tree but between /26 and /27. A route that matches the exact point, or has a subnet mask shorter than /26 or longer than /27, does not match the filter, as in Figure 6.20.

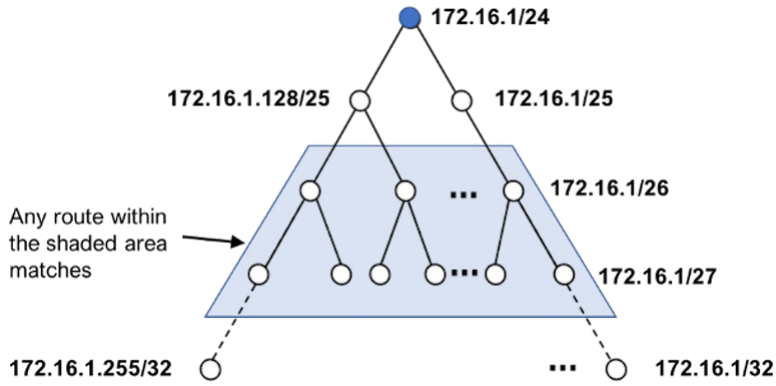


Figure 6.20 *172.16.1.0/24 prefix-length-range /26 - /27*

If the filter is `172.16.1/24 through 172.16.1.192/27`, the only routes that will match are: 172.16.1/24, 172.16.1.128/25, 172.16.1.192/26, and 172.16.1.192/27, as in Figure 6.21.

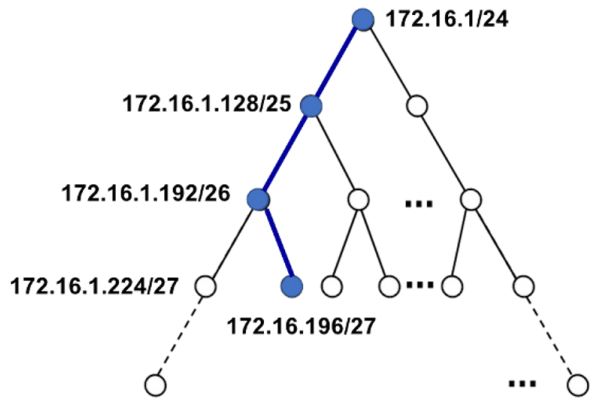


Figure 6.21 *172.16.1.0/24 through /27*

Let's now get into the details of how the four different filter options using this `<prefix > <subnet-mask-matching-type>` concept work, and what makes them different.

Prefix List

A prefix list creates a list of IP prefixes that you can then reference within a policy or within firewall filters.

You configured a prefix list under the [edit policy-options] hierarchy. Give the prefix list a name of your choice, which you will then use to reference the list from your policy, and then enter the prefixes that you want to match on. To show you how this works let's use Figure 6.22.

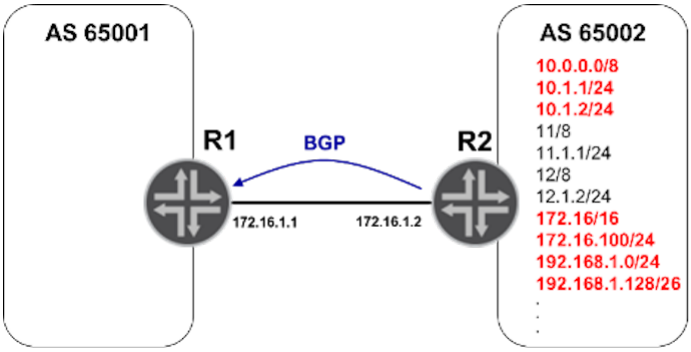


Figure 6.22 Prefix List Example

Say you are the administrator of R1. Your neighbor R2 is sending you all the routes that you see on the right within AS 65002. You want to configure a policy that rejects all the prefixes from the private address ranges (10/8, 172.16/12, and 192.168.0/16).

Currently, your routing table looks like this:

```
user@R1> show route protocol bgp terse
inet.0: 26 destinations, 26 routes (26 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 11.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 11.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 12.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 12.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 25.1.2.0/26        B 170        100            >172.16.1.2       65002 I
* 33.153.147.0/28    B 170        100            >172.16.1.2       65002 I
* 37.22.241.0/28     B 170        100            >172.16.1.2       65002 I
* 41.31.9.0/26       B 170        100            >172.16.1.2       65002 I
* 87.22.24.0/30      B 170        100            >172.16.1.2       65002 I
* 100.1.2.0/24       B 170        100            >172.16.1.2       65002 I
* 140.166.118.0/30   B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 191.223.181.0/25   B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
* 198.248.152.32/30  B 170        100            >172.16.1.2       65002 I
* 201.1.2.0/30       B 170        100            >172.16.1.2       65002 I
* 210.71.246.0/26    B 170        100            >172.16.1.2       65002 I
```

```
user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
* 10.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
```

As you can see, there are multiple routes within the private ranges that you need to remove.

To simplify your configuration you decide to use prefix-lists, so you create a prefix-list named REJECT-RFC1918, that matches on the private address ranges::

```
[edit policy-options prefix-list REJECT-RFC1918]
user@R1# show | display set relative
set 10.0.0.0/8
set 172.16.0.0/12
set 192.168.0.0/16
```

Then you create the policy that you will later apply to BGP.

You first check how to apply the prefix list that you created:

```
[edit policy-options policy-statement REJECT-BGP-RFC1918]
user@R1# set term 1 from p?
Possible completions:
+ policy               Name of policy to evaluate
  preference           Preference value
  preference2          Preference value 2
> prefix-list          List of prefix-lists of routes to match
> prefix-list-filter   List of prefix-list-filters to match
+ protocol             Protocol from which route was learned
```

And configure the policy as:

```
[edit policy-options policy-statement REJECT-BGP-RFC1918]
user@R1# show | display set relative
set term 1 from protocol bgp
set term 1 from prefix-list REJECT-RFC1918
set term 1 then reject
set term 2 from protocol bgp
set term 2 then accept
```

You remember to apply the policy as an IMPORT policy as you want to filter routes that you are receiving from your neighbor:

```
[edit protocols bgp group EBGP]
user@R1# set import REJECT-BGP-RFC1918
```

You commit your configuration and check the routing table again, only to discover something unexpected:

```
user@R1> show route protocol bgp terse
inet.0: 26 destinations, 26 routes (25 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
```

```
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 11.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 11.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 12.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 12.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 25.1.2.0/26        B 170        100            >172.16.1.2       65002 I
* 33.153.147.0/28    B 170        100            >172.16.1.2       65002 I
* 37.22.241.0/28     B 170        100            >172.16.1.2       65002 I
* 41.31.9.0/26       B 170        100            >172.16.1.2       65002 I
* 87.22.24.0/30      B 170        100            >172.16.1.2       65002 I
* 100.1.2.0/24       B 170        100            >172.16.1.2       65002 I
* 140.166.118.0/30   B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 191.223.181.0/25   B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
* 198.248.152.32/30  B 170        100            >172.16.1.2       65002 I
* 201.1.2.0/30       B 170        100            >172.16.1.2       65002 I
* 210.71.246.0/26    B 170        100            >172.16.1.2       65002 I

user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
```

Before applying the policy you were seeing these routes:

```
user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
* 10.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
```

And now you see these routes:

```
user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16      B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24    B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26   B 170        100            >172.16.1.2       65002 I
```

The only difference is that the 10.0.0.0/8 route is no longer there.

Though actually you can see it if you do:

```
user@R1> show route protocol bgp hidden
inet.0: 26 destinations, 26 routes (25 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/8          [BGP ] 00:45:03, localpref 100
```

```
      AS path: 65002 I
    > to 172.16.1.2 via fe-0/0/0.0
```

NOTE The route is hidden because it was rejected by the policy.

So your prefix-list was able to match 10.0.0.0/8, but not 10.1.1.0/24 or 10.1.2.0/24. Why?

Let's check the prefix-list and policy again:

```
[edit policy-options prefix-list REJECT-RFC1918]
user@R1# show
10.0.0.0/8;
172.16.0.0/12;
192.168.0.0/16;

[edit policy-options policy-statement REJECT-BGP-RFC1918 term 1]
user@R1# show | display set relative
set from protocol bgp
set from prefix-list REJECT-RFC1918
set then reject
```

At no time did you configure any of the prefix-length matching types we discussed before, did you?

That is because when you reference a prefix-list in a policy *using from prefix-list,* the subnet mask length matching type is *always EXACT*. Thus, our policy is matching on exactly 10/8, 172.16/12, and 192.168/16. The only route that matches is 10.0.0.0/8.

How do you complete the task then? How do you get rid of 10.1.1.0/24 and all the other prefixes? That's part of our next option: *prefix-list-filters*.

Prefix-list-filters

When you use prefix list filters you create a prefix list exactly the same way as before: you configured it under the [edit policy-options] hierarchy. You give the prefix list a name of your choice, which you will then use to reference from your policy, and then list the prefixes that you want to match on.

The difference here is the way you reference or call the `prefix-list` from the `routing-policy`. In this case, instead of using from `prefix-list` we are going to use from `prefix-list-filter`, which allows us to change the subnet mask matching type:

```
[edit policy-options policy-statement REJECT-BGP-RFC1918]
user@R1# set from pr?
Possible completions:
  preference           Preference value
  preference2          Preference value 2
> prefix-list          List of prefix-lists of routes to match
> prefix-list-filter   List of prefix-list-filters to match
+ protocol             Protocol from which route was learned
```

```
[edit policy-options policy-statement REJECT-BGP-RFC1918]
user@R1# set from prefix-list-filter ?
Possible completions:
  <list_name>          Name of prefix-list of routes to match
  exact                Exactly match the prefix length
  longer               Mask is greater than the prefix length
  orlonger             Mask is greater than or equal to the prefix length
```

You can see that the options for `prefix-list-filter` are only `exact`, `longer`, `orlonger`. Thus, they provide more options than `prefix-list`, but not all the options.

Okay, back to our scenario: we are trying to reject the following routes:

```
user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
* 10.0.0.0/8          B 170        100            >172.16.1.2       65002 I
* 10.1.1.0/24         B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24         B 170        100            >172.16.1.2       65002 I
* 172.16.0.0/16       B 170        100            >172.16.1.2       65002 I
* 172.16.100.0/24     B 170        100            >172.16.1.2       65002 I
* 192.168.1.0/24      B 170        100            >172.16.1.2       65002 I
* 192.168.1.128/26    B 170        100            >172.16.1.2       65002 I
```

You can use the same prefix list created in the previous example.:

```
[edit policy-options prefix-list REJECT-RFC1918]
user@R1# show| display set relative
set 10.0.0.0/8
set 172.16.0.0/12
set 192.168.0.0/16
```

But to filter 10.0.0.0/8, 10.1.1.0/24, and 10.1.2.0/24 we are going to use a `prefix-list-filter` and add `orlonger`. Remember, `orlonger` matches the prefix, and any subnet of the prefix.

Let's modify the policy the following way:

```
[edit policy-options policy-statement REJECT-BGP-RFC1918 term 1 from]
user@R1# delete prefix-list REJECT-RFC1918

[edit policy-options policy-statement REJECT-BGP-RFC1918 term 1 from]
user@R1# set prefix-list-filter REJECT-RFC1918 orlonger
```

And after committing, check the routing table again:

```
user@R1> show route protocol bgp terse
inet.0: 28 destinations, 28 routes (21 active, 0 holddown, 7 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 11.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 11.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 12.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 12.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 25.1.2.0/26        B 170        100            >172.16.1.2       65002 I
* 33.153.147.0/28    B 170        100            >172.16.1.2       65002 I
* 37.22.241.0/28     B 170        100            >172.16.1.2       65002 I
* 41.31.9.0/26       B 170        100            >172.16.1.2       65002 I
* 87.22.24.0/30      B 170        100            >172.16.1.2       65002 I
* 100.1.2.0/24       B 170        100            >172.16.1.2       65002 I
```

```
* 140.166.118.0/30   B 170        100            >172.16.1.2       65002 I
* 191.223.181.0/25   B 170        100            >172.16.1.2       65002 I
* 198.248.152.32/30  B 170        100            >172.16.1.2       65002 I
* 201.1.2.0/30       B 170        100            >172.16.1.2       65002 I
* 210.71.246.0/26    B 170        100            >172.16.1.2       65002 I
```

Notice that we now have a more hidden route:

```
user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
(no output)

user@R1>

user@R1> show route protocol bgp terse hidden | match  10\.| 192.168| 172.16.*.*/*
  10.0.0.0/8         B            100            >172.16.1.2       65002 I
  10.1.1.0/24        B            100            >172.16.1.2       65002 I
  10.1.2.0/24        B            100            >172.16.1.2       65002 I
  172.16.0.0/16      B            100            >172.16.1.2       65002 I
  172.16.100.0/24    B            100            >172.16.1.2       65002 I
  192.168.1.0/24     B            100            >172.16.1.2       65002 I
  192.168.1.128/26   B            100            >172.16.1.2       65002 I
```

We have achieved the objective! Now, someone might ask: *Do I always have to first create a prefix-list and reference it from the policy?* That's what it looks like at this point, right?

And, *what about the other options?* `Through`? `Upto`? Those are *not* available with `prefix-list-filters`.

Well. Time to talk about the third option: route filters.

Route Filters

Route filters are configured directly under a term within a policy, and are not reusable. While `prefix-list` are configured separately under `policy-options` and can be referenced from multiple policies or even terms within the same policy, `route-filters` do not even a have a name that can be referenced. To configure a route filter you go straight to the policy:

```
[edit policy-options policy-statement ROUTE-FILTER-POLICY]
user@R1# set term 1 from route-filter ?
Possible completions:
  <address>            IP address or hostname
  address-mask         Mask applied to prefix address
  exact                Exactly match the prefix length
  longer               Mask is greater than the prefix length
  orlonger             Mask is greater than or equal to the prefix length
  prefix-length-range  Mask falls between two prefix lengths
  through              Route falls between two prefixes
  upto                 Mask falls between two prefix lengths
```

As you can see, there is no `name`, but all the options we described before, are now available. So let's try to resolve the same scenario we were discussing before but now using `route-filters` instead or prefix-lists. First create this new policy:

```
[edit policy-options policy-statement ROUTE-FILTER-POLICY term REJECT-RFC1918]
user@R1# show | display set relative
set from protocol bgp
set from route-filter 10.0.0.0/8 orlonger
set from route-filter 172.16.0.0/12 orlonger
set from route-filter 192.168.0.0/16 orlonger
set term REJECT-RFC1918 then reject
```

Notice the use of `orlonger` because we want it to match the prefix, and any subnet of the prefix. And now apply the policy to BGP:

```
[edit protocols bgp group EBGP]
user@R1# set import ROUTE-FILTER-POLICY
```

After committing we check the routing table. The routes that are supposed to be rejected should not be listed:

```
user@R1> show route protocol bgp terse
inet.0: 28 destinations, 28 routes (21 active, 0 holddown, 7 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 11.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 11.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 12.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 12.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 25.1.2.0/26        B 170        100            >172.16.1.2       65002 I
* 33.153.147.0/28    B 170        100            >172.16.1.2       65002 I
* 37.22.241.0/28     B 170        100            >172.16.1.2       65002 I
* 41.31.9.0/26       B 170        100            >172.16.1.2       65002 I
* 87.22.24.0/30      B 170        100            >172.16.1.2       65002 I
* 100.1.2.0/24       B 170        100            >172.16.1.2       65002 I
* 140.166.118.0/30   B 170        100            >172.16.1.2       65002 I
* 191.223.181.0/25   B 170        100            >172.16.1.2       65002 I
* 198.248.152.32/30  B 170        100            >172.16.1.2       65002 I
* 201.1.2.0/30       B 170        100            >172.16.1.2       65002 I
* 210.71.246.0/26    B 170        100            >172.16.1.2       65002 I

user@R1> show route protocol bgp terse | match  10\.| 192.168| 172.16.*.*/*
(no output)

user@R1>
```

Just as predicted.

Now, one thing that you might have realized is that there is more than one route filter within the same term in a policy. The example we just reviewed is actually pretty straightforward.

There is a `route-filter` for each range of private addresses. `Route-filter 10.0.0.0/8 orlonger` will work for any route that begins with 10; `route-filter 172.16/12 orlonger` will be used for any route that begins with `172.00010000`, and so on.

But what if our neighbor is sending us all the routes shown in Figure 6.23 but we only care about 10.1.1/24.

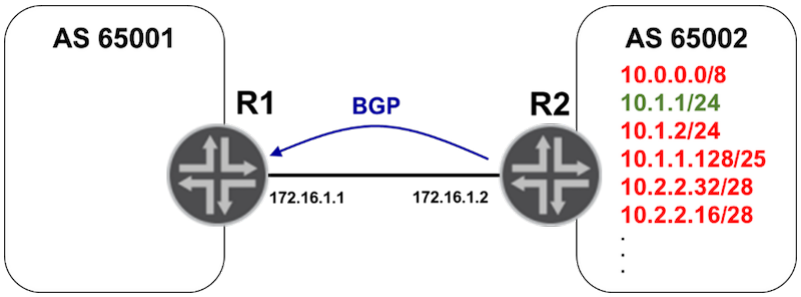


Figure 6.23 Route Filter 10.1.1/24

Since we have not applied a policy yet, you can see all the routes are installed in the routing table:

```
user@R1> show route protocol bgp terse
inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.0.0.0/8         B 170        100            >172.16.1.2       65002 I
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        100            >172.16.1.2       65002 I
* 10.2.2.16/28       B 170        100            >172.16.1.2       65002 I
* 10.2.2.32/28       B 170        100            >172.16.1.2       65002 I
* 10.1.1.128/25      B 170        100            >172.16.1.2       65002 I
```

We configure a new policy named ROUTE-FILTER-SOME-SUBNETS and apply it as an import policy to BGP:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
[edit  protocols bgp group EBGP]
user@R1# set import ROUTE-FILTER-SOME-SUBNETS
```

The routing table now shows 10.1.2/24 as an active route, and has four hidden routes:

```
user@R1> show route protocol bgp terse
inet.0: 11 destinations, 11 routes (7 active, 0 holddown, 4 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
* 10.1.1.128/25      B 170        100            >172.16.1.2       65002 I
```

But, surprise! 10.1.1.128/25 is also accepted and active. Why? How come the router is correctly deciding that 10/8, 10.1.2.0/24, 10.2.2.16/28, and 10.2.2.32/28 should be rejected, while 10.1.1/24 should be accepted, but it's failing to reject 10.1.1.128/25?

One of the things that confuses people and makes them question results like this is that our first instinct when we look at a policy with multiple route filters is to say: it's going to go in order, and the first route filter that matches the route is going to be used.

We just saw that this is not the case. If route filters were evaluated in order then, *all* the routes that we are receiving from R2 would be rejected, even 10.1.1.0/24.

But it turns out that even if we swapped the two filters, the results will still be the same:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS then reject
```

Our instinct also tells us that if the route does not match one route filter, the router checks the other router filter. Following this reasoning, you would think that 10.1.1.128/25 does not match `route-filter 10.1.1.0/24 exact` because the mask is not exactly 24, but it does match the `route-filter 10/8 orlonger` and therefore should be rejected. But that's not the case either!

Remember the two step process we talked about to check if a route matches a filter with a particular subnet match type?

Here it is again:

Step 1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask) Step 2: Compare the number of bits in route's subnet mask with the number of bits in the router-filter's subnet mask according to the subnet mask match type configured in the filter.

That's exactly what the router is going to do, but as part of Step 1, it will also figure out which one of the route filters is *the longest match*, and only proceed to Step 2 for that route filter.

Going back to the example, here is our policy again:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

Let's first analyze the decision process for *10.1.1/24*:

Step 1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask) AND find the longest match.

For this step, we're going to pretend that the match type is not there and find the longest match:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

You don't need to do much binary math to realize that 10.1.1.0/24 matches the first route filter, which specifies that the route should match 10.0.0.0 in the first 8 bits, but also matches the second route filter that specifies that the route should match 10.1.1.0 in the first 24 bits.

The *longest match* is the second route filter, and the route satisfies the criteria of step 1.

At this point we will pretend that the first route filter does not exist anymore, and proceed to step 2, but *only for the second route-filter*:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

Step 2: Compare the number of bits in the route's subnet mask with the number of bits in the router-filter's subnet mask according to the subnet mask match type configured in the filter.

The second route filter specifies that the subnet mask has to be exactly 24 bits long. The route that we are currently evaluating (10.1.1.0/24) has a subnet mask that is exactly /24. Thus the subnet mask criteria are also met.

Since this route filter has its own action (`accept`), the route is accepted and the policy processing stops. The route is now in the routing table!

NOTE We'll talk more about why the processing stops in the *Policies Actions* section of this chapter.

Let's now analyze the decision process for *10.2.2.16/28*.

Step 1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask) AND find the longest match.

Again, pretend that the match type is not there and find the longest match:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

You can also quickly see that 10.2.2.16/28 matches the first route filter, which specifies that the route should match 10.0.0.0 in the first 8 bits, but does *not* match the second route filter, which specifies that the route should match 10.1.1.0 in the first 24 bits.

Therefore, for this route, the longest match is the first route filter.

Once again pretend that the second route filter does not exist, and proceed to step 2, but only for the first route-filter:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

Step 2: Compare the number of bits in route's subnet mask with the number of bits in the router-filter's subnet mask according to the subnet mask match type configured in the filter.

The first route-filter specifies that the subnet mask has to be 8 bits or longer. The route that we are currently evaluating (10.2.2.16/28) has a subnet mask that is longer than /8. Thus, the subnet mask criteria are also met.

This route filter does not have its own action. Thus the term's action reject is applied in this case.

Let's finally analyze the decision process for 10.1.1.128/25, the route that caused us to scratch our heads:

Step 1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask) AND find the longest match.

Once again, we pretend that the match type is not there, and find the longest match:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

Of course, 10.1.1.128/25 matches the first route-filter which specifies that the route

should match 10.0.0.0 in the first 8 bits, but it also matches the second route filter that specifies that the route should match 10.1.1.0 in the first 24 bits.

Thus, for this route, the longest match is the second route filter.

As you did before, pretend that the first route-filter does not exist anymore, and proceed to step 2, but only for the first route-filter:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS then reject
```

Step 2: Compare the number of bits in the route's subnet mask with the number of bits in the router-filter's subnet mask according to the subnet mask match type configured in the filter.

The second route-filter specifies that the subnet mask has to be exactly /24. The route that we are currently evaluating (10.1.1.128/25) has a subnet mask that is /25. Thus the subnet mask criteria are not met.

Because the route does not end up matching the router is going to go back to the term and compare the route against the other route filter. Once the longest match has been found, that's it! It's literally as if any other route-filter within the term has been erased.

At this point, the router has determined that the route does NOT match term ACCEPT_SOME-SUBNETS, and that there are no more terms to evaluate, thus the route does NOT match the policy at all. Then what happens? The default policy for the protocol will be applied.

As we will see later, each routing protocol has its own default policies. For now, we are just going to say that the default BGP import policy is to accept ALL valid BGP routes. That is why we were seeing every route received from R2, before applying our policy, and why the route for 10.1.1.128/25 is being accepted, even after applying our policy.

To prove it, let's add a second term to the policy, commit, and check the routing table again:

```
[edit policy-options policy-statement ROUTE-FILTER-SOME-SUBNETS]
user@R1# show | display set relative
set term ACCEPT_SOME-SUBNETS from route-filter 10.1.1.0/24 exact accept
set term ACCEPT_SOME-SUBNETS from route-filter 10.0.0.0/8 orlonger
set term ACCEPT_SOME-SUBNETS then reject
set term REJECT-10.1.1.128 from route-filter 10.1.1.128/25 exact
set term REJECT-10.1.1.128 then reject

user@R1> show route protocol bgp terse
inet.0: 12 destinations, 12 routes (7 active, 0 holddown, 5 hidden)
```

```
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.0/24        B 170        100            >172.16.1.2      65002 I
```

As you can see the route for `10.1.1.128/25` is now gone.

Route-filter-list

The `route-filter-list` option was added in Junos Release 16.1 on M Series, MX Series, and T Series.

As you probably guessed, it combines the idea of having a reusable list, like prefix-lists, with having all the different subnet mask matching options that we have available with route-filters.

Let's use the same example we were just looking at, using `router-filter-lists` as shown in Figure 6.24.

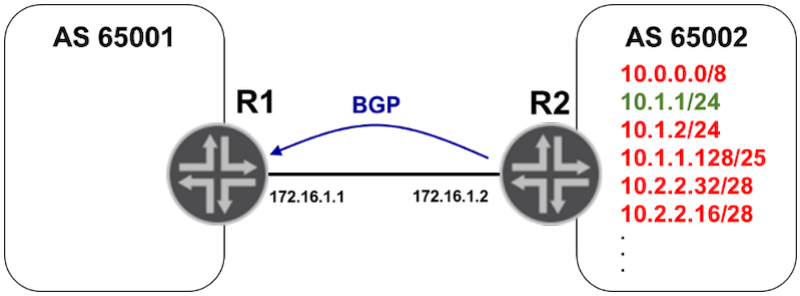


Figure 6.24 Using Router-Filter-Lists

As before, we want to allow only 10.1.1/24 and reject all the other 10/8 routes. Here is what our routing table looks like before:

```
user@R1> show route protocol bgp terse
inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
A V Destination        P Prf   Metric 1   Metric 2  Next hop        AS path
* ? 10.0.0.0/8         B 170        100                           65002 I
  unverified                                      >172.16.1.2
* ? 10.1.1.0/24        B 170        100                           65002 I
  unverified                                      >172.16.1.2
* ? 10.1.1.128/25      B 170        100                           65002 I
  unverified                                      >172.16.1.2
* ? 10.1.2.0/24        B 170        100                           65002 I
  unverified                                      >172.16.1.2
* ? 10.2.2.16/28       B 170        100                           65002 I
  unverified                                      >172.16.1.2
* ? 10.2.2.32/28       B 170        100                           65002 I
  unverified                                      >172.16.1.2
```

NOTE The output looks slightly different because the router is running a different Junos version.

Create a new (very simple) policy, that references a route-filter-list:

```
[edit policy-options policy-statement ROUTE-FILTER-LIST-POLICY]
jcluser@vMX-addr-0# show | display set relative
set term ROUTE-FILTER-LIST from route-filter-list ROUTE-FILTER-LIST
set term ROUTE-FILTER-LIST then accept
```

And create the route-filter-list where the magic is going to happen:

```
[edit policy-options route-filter-list ROUTE-FILTER-LIST]
jcluser@vMX-addr-0# show | display set relative
set 10.0.0.0/8 orlonger reject
set 10.1.1.0/24 exact
```

The processing will be the same as before: 1) check the number of bits (find the longest match) and 2) check the subnet mask match type.

The router will find the longest match among all the route-filters within the route-filter-list referenced within the policy.

Notice that the first entry in the route-filter-list has its own action of reject, AND specifies routes that match 10.0.0.0 in the first 8 bits. All the routes that we are receiving will match this entry and will be rejected.

However, 10.1.1.0/24 matches the second entry in the route-filter-list as well as specifying which routes match 10.1.1.0 in the first 24 bits. Thus the second `route-filter` within the `route-filter-list` is the longest match for this route.

For this second route-filter the subnet mask math type is exact. Our route meets the criteria. This second entry does not have its own action, therefore it just returns a TRUE value (yes, the route does match), and the action specified under the policy (accept) will be applied.

After applying the policy, we see the expected result:

```
[edit  protocols bgp]
jcluser@vMX-addr-0# set group EBGP import ROUTE-FILTER-LIST-POLICY

user@R1> show route protocol bgp terse
inet.0: 10 destinations, 10 routes (5 active, 0 holddown, 5 hidden)
+ = Active Route, - = Last Active, * = Both
A V Destination        P Prf   Metric 1   Metric 2  Next hop        AS path
* ? 10.1.1.128/25      B 170        100                           65002 I
  unverified                                      >172.16.1.2
```

Prefix Matching Options Summary

In case you are now wondering how to keep track of all these options, and how to remember which option is which, how they are different, and so on, Table 6.2 provides some clarity.

Table 6.2 Options Summary

Filter Type	Characteristics

route-filter	• Configured directly under the policy/term and has no name that can be used to reference it.
	• Cannot be reused by other policies/terms. Only applies to the policy/term where the route filter is configured.
	• Provides the match type options (exact, longer, orlonger, through, upto, prefix-length-range)
	• Allows defining an action directly after the from route-filter statement.
	• Example:
	`[edit policy-options policy-statement EXAMPLE]`
	`user@R1# show | display set relative`
	`set term ROUTE-FILTER from route-filter 10.1.1.0/24 orlonger accept`

prefix-list list-name> <prefix-	• A prefix-list configured under policy option is referenced by name using the from statement within one or more routing-policies/term.
	• The referenced prefix-list can be reused (can be referenced by multiple policies and terms within a policy. Can also be referenced by firewall filters.
	• The referenced prefix-list does not have any math type option. When references using from prefix-list it is ALWAYS EXACT. Other options are NOT available.
	• Does NOT allows defining an action directly after the from prefix-list.
	• Example:
	`[edit policy-options policy-statement EXAMPLE]`
	`user@R1# show | display set relative`
	`set term PREFIX-LIST from prefix-list EXAMPLE-PREFIX-LIST`
	
	`[edit policy-options prefix-list EXAMPLE-PREFIX-LIST]`
	`user@R1# show | display set relative`
	`set 10.0.0.0/8`
	`set 172.16.0.0/12`
	`set 192.168.0.0/16`

prefix-list-filter <prefix-list-name> [longer|orlonger|exact]	• A prefix-list configured under policy option is referenced by name using the from statement within one or more routing-policies/term.
	• The referenced prefix-list can be reused (can be referenced by multiple policies and terms within a policy. Can also be referenced by firewall filters.
	• The referenced prefix-list does not have any math type option. When references used from prefix-list-filter you can configure the match types: exact, longer, and orlonger.
	• Allows defining an action directly after the from route-filter statement.
	• Example:
	`[edit policy-options policy-statement EXAMPLE]`
	`user@R1# show | display set relative`
	`set term PREFIX-LIST from prefix-list-filter EXAMPLE-PREFIX-LIST orlonger accept`
	
	`[edit policy-options prefix-list EXAMPLE-PREFIX-LIST]`
	`user@R1# show | display set relative`
	`set 10.0.0.0/8`
	`set 172.16.0.0/12`
	`set 192.168.0.0/16`

route-filter-list <route-filter-list-name>	• Introduced in Junos Release 16.1 on M Series, MX Series, and T Series, and not included in the current IJOS version.
	• A route-filter-list configured under policy option is referenced using the from statement withing one or more routing-policies/term.
	• The referenced route-filter-list can be reused (can be referenced by multiple policies and terms within a policy. Can also be referenced by firewall filters.
	• The referenced route-filter-list provides the match type options (exact, longer, orlonger, through, upto, prefix-length-range).
	• Example:
	`[edit policy-options policy-statement EXAMPLE]`
	`user@R1# show | display set relative`
	`set term ROUTE-FILTER-LIST from route-filter-list ROUTE-FILTER-LIST-EXAMPLE`
	
	`[edit policy-options route-filter-list ROUTE-FILTER-LIST-EXAMPLE]`
	`user@R1# show| display set relative`
	`set 10.1.0.0/16 exact accept`
	`set 10.1.0.0/16 longer reject`
	`set 172.16.0.0/12 orlonger`

MORE? To learn even more, go to the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/policy-configuring-route-lists-for-use-in-routing-policy-match-conditions.html.

Actions – The Then Statement

We will now look at the different actions that a router can take when a route matches a term within a policy. So far, we have mentioned reject and accept which are quite self-explanatory. But there are many other actions that we implement.

First thing you need to know is that there are three types of actions:

- Terminating Actions
- Flow Control Actions
- Modifying Actions

Terminating Actions

Terminating actions do just that: they terminate the processing of a policy. There are only two such actions: ACCEPT, or REJECT, and you have seen examples of these actions already.

If you are applying a policy as an import policy, an ACCEPT action means the route will be installed in the routing table, and can become active, while a REJECT means the route cannot be used, and becomes a hidden route.

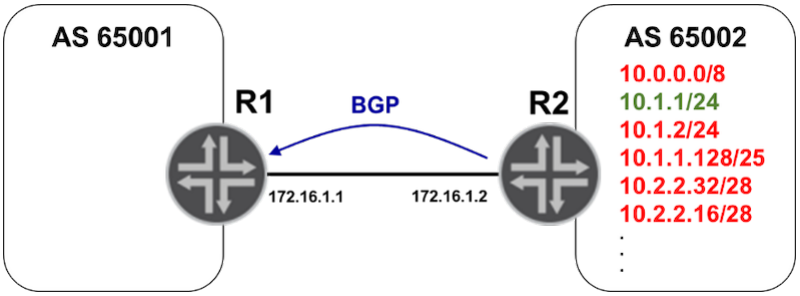
If you are applying a policy as an export policy to BGP, for example, an ACCEPT action means the route will be advertised to your BGP neighbor(s), while a REJECT means the route will not be advertised.

But what exactly does *terminate processing"* mean?

Remember that when a route is evaluated, terms are checked sequentially from top to bottom. When a match is found, the router applies the actions specified within the matching term, and if one of the actions is terminating, no more processing is required. In other words, the actions in the term are applied, the route is either accepted or rejected, and no more terms or policies are checked. If a route matches the criteria within a term, but this term does *not* have a terminating action, which is a valid situation, processing of the policy continues.

Consider this example: R1 is receiving routes from R2 for all the prefixes listed on the right side as shown in Figure 6.25.

Figure 6.25 R1 Receiving Routes



You only want to accept 10.1.1/24, but also want to make some changes to the route (modifying actions covered in more detail later) before the route is installed in the routing table.

This is how the route looks like before we apply the new policy that we will create in a minute. Pay attention to the values of Metric 1 (100), and Metric 2 (none):

```
user@R1> show route protocol bgp terse 10.1.1/24 exact
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.0/24        B 170        100            >172.16.1.2       65002 I
```

We create a new policy that looks like this:

```
[edit policy-options policy-statement NOT-TERMINATING]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 then local-preference 200

set term 2 then metric 300
set term 2 then accept

set term 3 then reject
```

Notice that term 1 changes Metric 1 (`local-preference`) to 200, and term 2 changes Metric 2 to 300. Again, we are making changes (modifying actions) here, just to demonstrate which term is matching and accepting/rejecting the routes, but we will cover these later.

We apply the policy and after committing check the routing table again:

```
[edit  protocols bgp group EBGP]
user@R1# set import NOT-TERMINATING

user@R1> show route protocol bgp terse 10.1.1/24 exact
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.0/24        B 170        200        300 >172.16.1.2       65002 I
```

The result shows that when the 10.1.1.0/24 route is evaluated against the first term it matches the router-filter, thus the local-preference (Metric 1) is set to `200`. Because the term does not have a terminating action, the route is then evaluated against the next term, which does not have a from statement (all routes match).

When the route is evaluated against term 2 Metric2 is set to `300` AND the route is accepted. Processing stops at this point and the route is placed in the routing table and becomes active. If term 2 did not have the terminating action accept, the route will then be evaluated against term 3 and would be rejected.

Let's try deactivate the action within term 2:

```
[edit policy-options policy-statement NOT-TERMINATING term 2 then]
user@R1# deactivate accept

user@R1> show route protocol bgp terse 10.1.1/24 exact hidden
inet.0: 12 destinations, 12 routes (6 active, 0 holddown, 6 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
  10.1.1.0/24        B            200        300 >172.16.1.2      65002 I
```

The route still gets the two metric values set but is rejected at the end by term 3. Therefore, the route has become a hidden (unusable) route.

Flow Control Actions

Flow control actions can be used to tell the router to jump to skip the rest of the actions within a term, or the rest of a policy.

There are two flow control actions available: NEXT-TERM, NEXT-POLICY, and as you can imagine, they mean: go to the next term, or go to the next policy.

As an example, let's say that we want to again only accept 10.1.1/24 and change its local preference to 500. We want to reject all other 10/8 routes.

NOTE: We are changing the local preference at this point just to demonstrate which term is matching and accepting the route.

This time we are going to use the `next-term` action to achieve the result. Here is how our new policy will look:

```
[edit policy-options policy-statement FLOW-CONTROL]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 from route-filter 10.0.0.0/8 orlonger next term
set term 1 then local-preference 500
set term 1 then accept
set term 2 then reject
```

What term 1 says is: If the route matches 10.1.1.0/24 exactly, set the `local-preference` to 500 and accept the route. Term 1 also says: if the route matches 10/8 `orlonger` (any other 10/8 route), skip the rest of the term and go to the next one.

After applying the policy here is what the routing table looks like:

```
user@R1> show route 10/8 protocol bgp terse
inet.0: 12 destinations, 12 routes (7 active, 0 holddown, 5 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.0/24        B 170        500            >172.16.1.2       65002 I

user@R1> show route 10/8 protocol bgp terse hidden
R1.inet.0: 12 destinations, 12 routes (7 active, 0 holddown, 5 hidden)
+ = Active Route, - = Last Active, * = Both
A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
  10.0.0.0/8         B            100            >172.16.1.2       65002 I
  10.1.1.128/25      B            100            >172.16.1.2       65002 I
  10.1.2.0/24        B            100            >172.16.1.2       65002 I
  10.2.2.16/28       B            100            >172.16.1.2       65002 I
  10.2.2.32/28       B            100            >172.16.1.2       65002 I
```

Notice that the routes that are being rejected by term 2 did not get their `local-preference` modified, as 10.1.1/24, which was accepted by term 1, did.

Modifying Actions

Modifying actions allow you to change the characteristics or attributes of routes. You can change, add, or remove attributes, depending on which routing protocols or which attribute you are working with and sometimes whether you are applying the policy as an import or xport policy. Some examples of actions that you can implement:

For OSPF:

- change the metric of routes that you are redistributing into OSPF
- change the external route type, or the preference value
- add/remove tags

For BGP:

- change the local-preference, the MED, the route origin, the AS-PATH
- add, replace, or remove communities

If you configure a policy that tries to modify an attribute that is not supported by the protocol you are applying the policy to, or the direction of the policy (import/export), the attribute is simply ignored. For example, consider Figure 6.26.

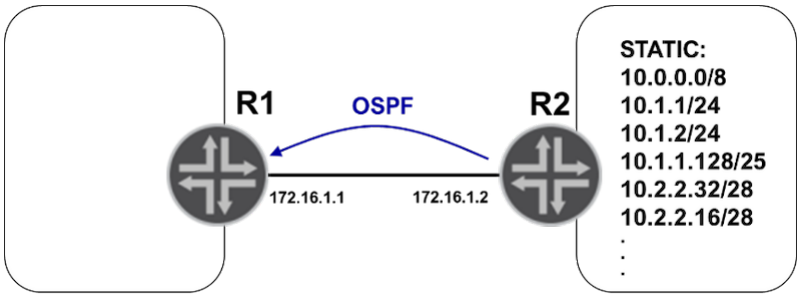


Figure 6.26 Modifying Attributes

We configured the following policy on R2:

```
[edit policy-options policy-statement STATIC]
user@R2# show | display set relative
set term 1 from protocol static
set term 1 then metric 20
set term 1 then accept
set term 1 then local-preference 200

[edit protocols ospf]
user@R2# show | display set relative
set export STATIC
set area 0.0.0.0 interface fe-0/0/1.0
```

And this policy on R1:

```
[edit policy-options policy-statement MODIFYING-ACTIONS]
user@R1# show | display set relative
set term 1 from protocol ospf
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 then metric 25
set term 1 then accept
set term 2 then reject

[edit protocols ospf]
user@R1# show | display set relative
set import MODIFYING-ACTIONS
set area 0.0.0.0 interface fe-0/0/0.0
```

We were able to commit the configuration on R2 even though the policy is asking the router to set the local-preference value, while matching on protocol OSPF. There is no local-preference field on the OSPF message, so the requested action is ignored.

We are also attempting to change the metric on both R1, and R2, but though the policy commits on both routers, the action is only implemented by R2 when the route is redistributed from static into OSPF.

We are not going to go over all the possible cases here, but as you learn more about routing protocols in Junos you will run into more specific situations. We want you to be aware from the beginning that there will be a lot of variations and caveats depending on the protocols, and what you are trying to achieve.

Another important concept is that *modifying actions are not terminating*, so don't expect that because a route matches and you are adding a tag for example, the route will automatically be accepted.

Consider the scenario shown in Figure 6.27.

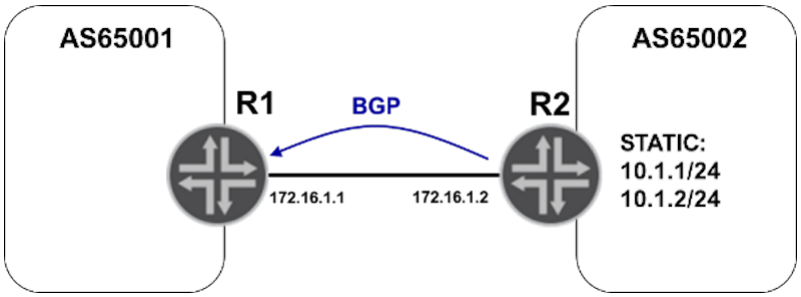


Figure 6.27 Modifying Actions Example 2

R2 is sending 10.1.1/24 and 10.1.2/24, and R1 has the following import policy:

```
[edit policy-options policy-statement ONLY/24]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 from route-filter 10.1.2.0/24 exact
set term 1 then local-preference 200

[edit  protocols bgp group EBGP]
user@R1# set import ONLY/24
```

When you check the routing table, your routes are there:

```
user@R1> show route protocol bgp terse
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
* 10.1.1.0/24        B 170        200            >172.16.1.2       65002 I
* 10.1.2.0/24        B 170        200            >172.16.1.2       65002 I
```

Do not make the mistake of assuming that the routes are being implicitly accepted just because you see the `local-preference` changed to `200`, indicating that the routes matched the term and there are no more terms in your policy. The default protocol action is always applied. Guess what the default import policy for BGP is? ACCEPT!

During policy evaluation, a copy of the actual route is used, and as it is processed through the policy it is modified as needed. However, the route is not copied to the routing table or advertised to a neighbor until the policy evaluation is complete, and again, that only happens when you hit a term with a terminating action.

Thus, when a route matches a term that has no then statement at all, or only has non-terminating actions, the router will continue processing the policy. In other words, it will evaluate the route against the rest of the policy until a match is found against a term with a terminating action.

NOTE The `then` statement is optional.

Many questions might pop up in your head at this point. For instance:

What happens if the route does *not* match any of the terms in my policy? Well, you could have applied a second policy, so the route will be evaluated against the next policy.

What if I don't have a second policy? Or what if my route does not match any of the terms in this second policy either? The default policy is applied!

And what is the default policy? Well, it depends! It depends on:

- Which protocol your policy(ies) is/are applied to
- Which direction your policy(ies) is/are applied in (export versus import)

In the next couple of sections we'll talk about applying more than one policy, and about the default policies.

Policy Chains

You can apply more than one policy to a protocol at the same time. When you do so, each policy will be evaluated as we already learned: term by term, in strict top to bottom order, until a match is found within a term with a terminating action.

If a route is evaluated against every term within the first policy applied to the protocol, without matching a term with a terminating action, the router moves to the next policy and the processing is the same within this second policy: term by term, and so forth. Figure 6.28 illustrates the idea.

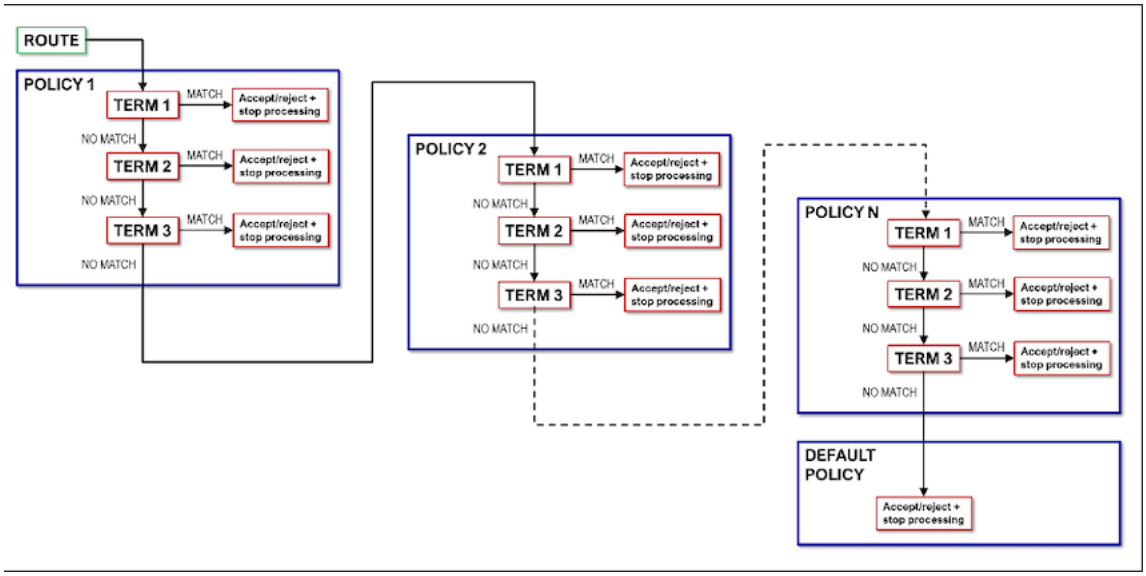


Figure 6.28 Policy Chains

The policies will also be evaluated in strict order, left to right, again until a match in a term with a terminating action is found.

Notice that at the end of this chain of policies there is a policy that is not user configurable: the protocol's default policy. This policy will always be evaluated last, when a match is not found, unless the last policy applied to the protocol has a term at the end such as:

```
[edit policy-options policy-statement LAST_POLICY]
user@R1# set term last then reject
```

That stops processing before the router gets to the default policy.

It is probably obvious at this point that just like the order of terms within a policy is important, the order of application of the policies is also important. Let's see how multiple policies can be applied, and how we can make sure that they are applied in the proper order.

You can apply more than one policy by simply entering the `import` or `export` command more than once as shown in this example:

```
[edit protocols bgp group EBGP]
user@R1# set import POLICY_3

[edit protocols bgp group EBGP]
user@R1# set import POLICY_1
```

```
[edit protocols bgp group EBGP]
user@R1# set import POLICY_2
```

Which results in this configuration:

```
[edit protocols bgp group EBGP]
user@R1# show | display set relative | match import
set import POLICY_3
set import POLICY_1
set import POLICY_2
```

Notice that just like the names of the terms have no sequencing significance, naming your policies with numbers does not matter either. You need to make sure you enter the policies in the correct order.

Sometimes you need to add an additional policy later on, and like terms, any new policy that you apply will be added at the bottom of the list:

```
[edit protocols bgp group EBGP]
user@R1# set import POLICY_NEW

[edit protocols bgp group EBGP]
user@R1# show | display set relative | match import
set type external
set import POLICY_3
set import POLICY_1
set import POLICY_2
set import POLICY_NEW
```

We can use the same trick we used before to reorder terms, to reorder policies:

```
[edit protocols bgp group EBGP]
user@R1# insert import POLICY_3 after POLICY_2

[edit protocols bgp group EBGP]
user@R1# insert import POLICY_NEW before POLICY_1

[edit protocols bgp group EBGP]
user@R1# show | display set relative
set type external
set import POLICY_NEW
set import POLICY_1
set import POLICY_2
set import POLICY_3
```

You can also enter the multiple policies using square brackets, like this:

```
[edit protocols bgp group EBGP]
user@R1# set import [ POLICY_NEW POLICY_1 POLICY_2 POLICY_3 ]
```

Which could make it easier for you to enter the policies in the correct order, if you need to make changes afterwards.

For example, if you wanted to add a new policy called POLICY_0, between POLICY_NEW and POLICY_1, you could do something like this:

```
[edit protocols bgp group EBGP]
user@R1# delete import

[edit protocols bgp group EBGP]
user@R1# set import [ POLICY_NEW POLICY_0 POLICY_1 POLICY_2 POLICY_3 ]
```

NOTE This option looks more like what you see when you check the configuration using the curly bracket format:

```
[edit protocols bgp group EBGP]
user@R1# show | match import
import [ POLICY_NEW POLICY_0 POLICY_1 POLICY_2 POLICY_3 ];
```

When you have multiple policies applied, at a given time, you could use the `flow-control` action `next` policy and skip the rest of the policy currently being evaluated.

For example, the policy that we wrote in the previous section:

```
[edit policy-options policy-statement FLOW-CONTROL]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 from route-filter 10.0.0.0/8 orlonger next term
set term 1 then local-preference 500
set term 1 then accept
set term 2 then reject

[edit protocols bgp group EBGP]
user@R1# set import FLOW-CONTROL
```

Could be replaced by these two policies:

```
[edit policy-options policy-statement FLOW_CONTROL_1]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 from route-filter 10.0.0.0/8 orlonger next policy
set term 1 then local-preference 500
set term 1 then accept

[edit policy-options policy-statement FLOW_CONTROL_2]
user@R1# show | display set relative
set term 2 then reject

[edit protocols bgp group EBGP]
user@R1# delete import

[edit protocols bgp group EBGP]
user@R1# show | display set
set protocols bgp group EBGP import FLOW_CONTROL_1
set protocols bgp group EBGP import FLOW_CONTROL_2
```

The last piece of this policy chain puzzle that you need to learn is the default policy. Let's check that now.

Default Policies

Each routing protocol has its own default routing policy and warning: they are very different, and it would help your future career dealing with Juniper devices to memorize them!

Let's start with a basic summary that will help you memorize things more easily.

Table 6.3 Protocol Default Policies

Protocol	Import Policy	Export Policy
BGP	Accept all BGP routes and import into inet.0	Accept all active BGP routes
OSPF	Accept all OSPF routes and import into inet.0	Reject everything (protocol floods by default)
IS-IS	Accept all IS-IS routes and import into inet.0	Reject everything (protocol floods by default)
RIP	Accept all RIP routes from explicitly configured neighbors and import into inet.0	Reject everything

But if you really want to know what all these default policies mean, you can look at the enhanced version in Table 6.4. And Figure 6.29 can help you visualize.

Table 6.4 Protocol Default Policies Detail

Protocol	Import Policy	Export Policy
BGP	BGP will validate that the routes received are good routes, that there is no strange attribute, or a loop. Thus the default policy is actually: All valid BGP routes are accepted and imported into the routing table. (could be inet.0, or another routing table like customer.inet0, inet6.0 depending on configuration)	Accept all active BGP routes in the routing table and advertise them to BGP neighbors following BGP rules. A BGP route that is present in the routing table but is not active, because the route was also learned from say static, will NOT be advertised by default. This default policy also means that no route redistribution happens by default. For instance, you don't advertise routes you learned from OSPF to your BGP neighbors by default.

OSPF	Accept all OSPF routes and import them? Yes, but that means all OSPF routes coming from SPF calculations (not from neighbors) and import them into the routing table. See the diagram after the table. Remember that OSPF is a Link State Routing Protocol, and instead of routes, routers exchange LSAs which will be accepted and placed in the LSDB. Distribution of LSAs 1 and 2 which describe the topology within an area will NOT be affected by policies. Import policies can be configured, but only to filter for external routes from LSAs type 5/7 or the distribution of LSAs type 3.	Rejects everything? What this really means is that routes in the routing table will be rejected, which translates into: NO redistribution by default !! The router by default will not take the BGP, static or any other route from the routing table and advertise them via OSPF! LSAs are still created for interfaces running OSPF, and sent out. (Distribution of LSAs 1 and 2 is NOT be affected by policies). An export policy would enable redistribution and create LSAs type 5/7. You could write your own export policy that simply says then reject, and OSPF will continue to work.
IS-IS	Accept all ISIS routes and import them? Yes, but like for OSPF, what this really means is: all routes coming from SPF calculations (not from neighbor) are accepted and imported into the routing table. LSPs sent by neighbors are accepted and placed in the LSDB, SPF calculates routes, routes. (Not affected by policies). Import policies could not be configured at all for ISIS up until 17.1. An import policy would allow filtering routes coming from the LSDB into the routing table.	Rejects everything? Yes, except for direct routes for interfaces running ISIS which are accepted. There is some redistribution of interface routes into ISIS under the hood, that would break, if you wrote your own export policy that simply says then reject. LSPs that are created for interfaces running ISIS, and sent out by default (can be affected by policy). An export policy in ISIS can control which prefixes are actually advertised (even for interfaces running ISIS, and also enables redistribution and create external routes – w/ narrow metrics). The main reason ISPs love ISIS. Like for OSPF, reject everything also means NO redistribution by default (for any other route).
RIP	RIP routes from explicitly configured neighbors are accepted and imported into the routing table.	Reject everything ? Yes, this time is really means that. NO RIP routes generated or forwarded by default. ALL routes are rejected = NO redistribution by default.

Also, for ISIS and OSPF, Figures 6.29 and 6.30 represent what is described in the table.

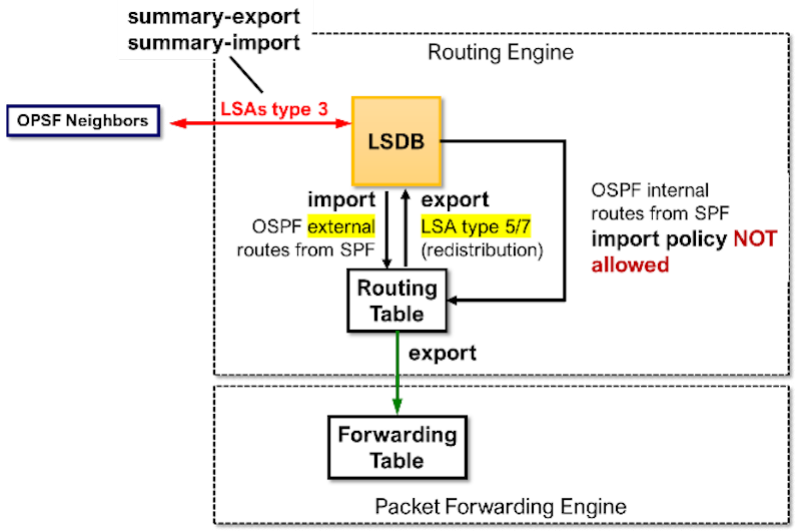


Figure 6.29 Protocol Default Policies Detail

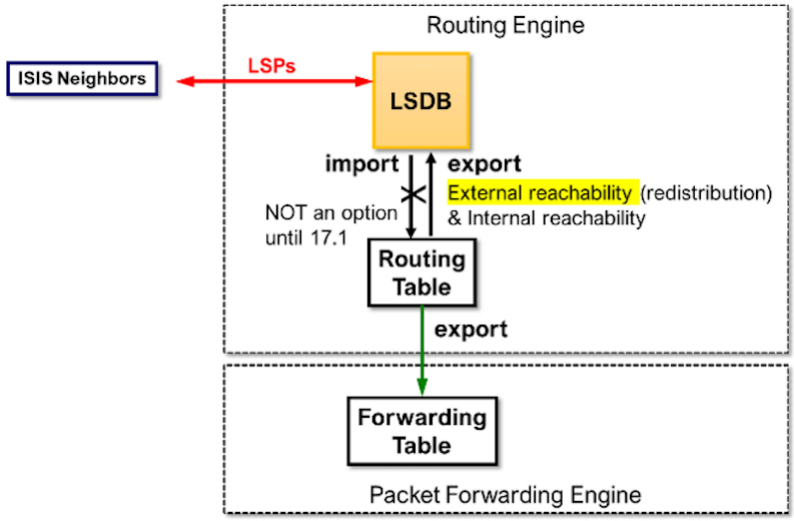


Figure 6.30 Route Redistribution Topology

If you still want to read more about routing policies default actions go to the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/concept/policy-routing-policies-actions-defaults.html.

Routing Policies Configuration and Monitoring Examples

We are now going to go over two complete examples and summarize the configuration and verification steps.

Route Redistribution

Figure 6.31a is our test scenario for this route redistribution example.

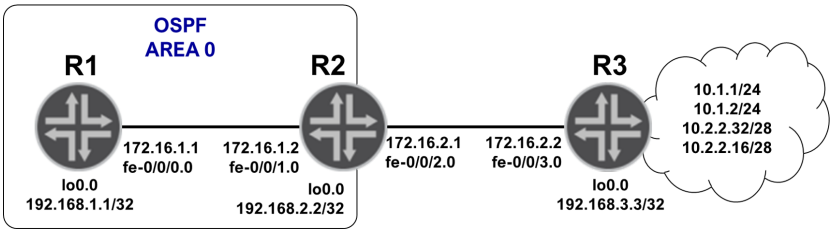


Figure 6.31a Test Scenario for Route Redistribution

You can see that R1 and R2 will be running single area OSPF, and will be advertising the addresses of their loopback interfaces. R2 will have static routes to reach the networks behind R3, as well as the loopback of R3. These routes need to be advertised to R1.

R3 needs to be configured with a static default route to reach the networks inside the OSPF domain.

STEPS

1) Configure the static routes on R2 and R3. Static routes are configured under the `routing-options` stanza using:

```
set static route <destination-prefix> next-hop <next-hop-address>
```

Remember that the next hop has to be reachable and that by default it has to be the address of the physical interface of a directly connected neighbor. On R2 configure:

```
[edit routing-options]
user@R2# set static route 10/8 next-hop 172.16.2.2

[edit routing-options]
user@R2# set static route 192.168.3.3/32 next-hop 172.16.2.2
```

On R3 configure:

```
[edit routing-options]
user@R1# set static route 0/0 next-hop 172.16.2.1
```

2) Verify the static routes were created. To verify the correct configuration of static routes you can check the routing table by typing:

```
show route protocol static
```

Check the static route on R2:

```
user@R2> show route protocol static
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/8          *[Static/5] 00:01:18
                     > to 172.16.2.2 via fe-0/0/2.0
192.168.3.3/32      *[Static/5] 00:01:18
                     > to 172.16.2.2 via fe-0/0/2.0
```

Check the static route on R3:

```
user@R3> show route protocol static
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
0.0.0.0/0           *[Static/5] 00:01:18
                             > to 172.16.2.1 via fe-0/0/3.0
```

As part of the verification check that:

- The routes were installed in inet.0 (the default IPv4 routing table).
- The routes are active (indicated with the * symbol) and were learned from static configuration.
- The subnet mask is correct.
- The next-hop and outbound interface are correct.
- And, just as refresher, we can also see that its preference value is 5, the (default value).

As this point we should be able to ping from R2 to the addresses behind R3:

```
user@R2> ping 192.168.3.3 rapid
PING 192.168.3.3 (192.168.3.3): 56 data bytes
!!!!!
--- 192.168.3.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.896/3.714/6.089/1.645 ms

user@R2> ping 10.1.1.1 rapid
PING 10.1.1.1 (10.1.1.1): 56 data bytes
!!!!!
--- 10.1.1.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.835/3.441/7.163/1.914 ms
```

3) Configure OSPF area 0 on R1 and R2, and make sure the loopback interface is included.

To configure OSPF go under the `edit protocols` stanza, with:

```
set ospf area <area_id> interface <interface-name>
```

Make sure you include the unit number when configuring the interface to make sure you are enabling OSPF on the correct interface.

Use the `passive` knob when you want an interface to be advertised but you do not want OSPF messages to be sent out of that interface or adjacencies to be created out of that interface.

Configure OSPF on R1 and R2 with:

```
[edit protocols]
user@R1# show | display set relative
set ospf area 0.0.0.0 interface fe-0/0/0.0
set ospf area 0.0.0.0 interface lo0.0 passive
```

NOTE Though really not required in this example, we are configuring `passive` for the loopback interface to demonstrate:

```
[edit protocols]
user@R2# show | display set relative
set ospf area 0.0.0.0 interface fe-0/0/1.0
set ospf area 0.0.0.0 interface lo0.0 passive
```

4) Verify the operation of OSPF – to check the operation of OSPF you can use the following commands:

```
show ospf interface
show ospf neighbor
show ospf database
show route protocols ospf
```

Check the OSPF interfaces on both routers with:

```
user@R1> show ospf interface
Interface  State  Area DR ID  BDR ID Nbrs
fe-0/0/0.0 BDR  0.0.0.0  192.168.2.2  192.168.1.1  1
lo0.1      DRother  0.0.0.0  0.0.0.0  0.0.0.0  0

user@R2> show ospf interface
Interface  State   Area DR ID  BDR ID Nbrs
fe-0/0/1.0 DR0.0.0.0  192.168.2.2  192.168.1.1  1
lo0.2      DRother  0.0.0.0  0.0.0.0  0.0.0.0  0
```

Here, check that:

- The correct interfaces were added to OSPF.
- The interfaces belong to the correct area (area 0).
- The routers are seeing each other, thus their neighbor count out of the fe interface is 1.
- R2 was elected the DR for the link between R1/R2 (election happens by default on Ethernet links).

Check the neighbor adjacencies status to confirm the router are neighbors:

```
user@R1> show ospf neighbor
Address    Interface  State     ID               Pri  Dead
172.16.1.2 fe-0/0/0.0 Full      192.168.2.2      128    39

user@R2> show ospf neighbor
Address    Interface  State     ID               Pri  Dead
172.16.1.1 fe-0/0/1.0 Full      192.168.1.1      128    39
```

You can verify here:

- The address of the neighbor and router ID of the neighbor.
- The interface the routers are learning about that neighbor.
- The state of the adjacency, which should be FULL.
- The priority of 128 that you see listed is the default priority for DR election. The election in this case was based on ID, as if the PRI were the same.

Check the OSPF database:

```
user@R1> show ospf database
    OSPF database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router  *192.168.1.1      192.168.1.1      0x80000003   936  0x22 0xbba4  48
Router   192.168.2.2      192.168.2.2      0x80000003   937  0x22 0xd286  48
Network  172.16.1.2       192.168.2.2      0x80000001   942  0x22 0xd063  32

user@R2> show ospf database
    OSPF database, Area 0.0.0.0
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Router   192.168.1.1      192.168.1.1      0x80000003   941  0x22 0xbba4  48
Router  *192.168.2.2      192.168.2.2      0x80000003   940  0x22 0xd286  48
Network *172.16.1.2       192.168.2.2      0x80000001   945  0x22 0xd063  32
```

Without getting into too much detail:

- You can see that the routers are exchanging LSA and are synchronized.
- The * marks the LSAs generated by the router where you are entering the command.

Check the routing table:

```
user@R1> show route protocol ospf
inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.2/32     *[OSPF/10] 00:19:43, metric 1
                    > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32       *[OSPF/10] 00:20:33, metric 1
                       MultiRecv

user@R1> show route protocol ospf
inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
192.168.1.1/32     *[OSPF/10] 00:19:43, metric 1
                    > to 172.16.1.1 via fe-0/0/1.0
224.0.0.5/32       *[OSPF/10] 00:20:33, metric 1
                       MultiRecv
```

You can check here:

- That the routers are learning about each other's loopback via OSPF.
- You can check the correct next hop and outbound interface and the metric, which is automatically calculated by the router using this formula (108/bandwidth).
- Also notice that the preference value is 10. The default preference for internal OSPF routes (routes that were learned from interfaces running OSPF, for example, loopback interface).

5) Create a routing policy on R2 to redistribute the static routes so that R1 can reach the addresses behind R2. Configure the routing policies under the policy-options stanza with:

```
set policy-statement <policy_name> term <term_name> from <matching_attribute>
set policy-statement <policy_name> term <term_name> then <action>
```

Remember that:

- You can have zero or more `from` statements, and when you have more than one, the route has to match all statements, to match the term.
- If you omit the `from` statement all routes match.
- If you enter more than one `from` statement of the same type (`from protocol ospf`, `from protocols bgp`) that is a logical OR. Though for prefix filters the router finds the longest match.
- If you enter more than one `from` statement of different types (from protocol OSPF, from neighbor 172.1.1.1) that is a logical AND.
- You can have zero or more actions:
  - If you don't configure an action, processing of the policy continues (check next term).
  - If you don't configure a terminating action, (accept/reject) processing of the policy continues.
  - Terms are checked top to bottom in strict order.
  - Policies are checked left to right in strict order.
  - If the route does not match any term within any user-configured policy, the default policy for the protocols is applied.

Create the policy using `route-filter` like this:

```
[edit policy-options policy-statement redistribute-static]
user@R1# show | display set relative
set term 1 from protocol static
set term 1 from route-filter 10.0.0.0/8 exact
set term 1 from route-filter 192.168.3.3/32 exact
set term 1 then accept
```

A route needs to math protocol static AND one of the two route-filters to match term 1 and be accepted.

6) We apply the policy under the `edit protocols <protocol>` stanza with:

```
set export <policy_name>
```

Or:

```
set import <policy_name>
```

Remember: the router imports and exports routes into and out of the routing table. It is always from the point of view of the routing table.

In our scenario:

- How do you determine which protocol you need to apply the policy to?
- Do you configure it as an import policy or an export policy?

R2 needs to take routes that it already knows about and *are present in the routing table* (the static routes) and send them to its OSPF neighbor. So, we are taking the static routes already in the routing table, which means EXPORT, and we are sending them to our OSPF neighbor, that means configure the policy under OSPF:

```
[edit protocols ospf]
user@R1# set export redistribute-static
```

7) Verify that the policy is working.

There are different ways you can do this, and that will depend on which routing protocol you are working with.

Since we are using OSPF, you know that information is sent using LSAs. In this case, the LSAs will be type 5 (external LSAs) which are created when you redistribute routes into OSPF. Thus what you need to do is check the OSPF database looking for external LSAs:

```
show ospf database | match Extern
```

You can enter this command on the router sending the LSA, and on the router receiving the LSA.

We can also check the routing table on the receiving side:

```
show route protocol ospf
```

If you were using BGP instead, information is sent using BGP updates (basically routes). In that case you could use:

```
show route advertising-protocol bgp <neighbor_address>
```

On the router advertising the routes:

```
show route receive-protocol bgp <neighbor_address>
```

On the router receiving the routes. Check the database on both R1 and R2:

```
user@R1> show ospf database | match Extern
Extern     10.0.0.0 192.168.2.2  0x80000001    35  0x22 0x58e2  36
Extern     192.168.3.3  192.168.2.2  0x80000001    35  0x22 0xe9eb  36

user@R2> show ospf database | match Extern
Extern     *10.0.0.0  192.168.2.2  0x80000001    37  0x22 0x58e2  36
Extern     *192.168.3.3 192.168.2.2  0x80000001    37  0x22 0xe9eb  36
```

What we are checking for:

- The presence of the Extern LSAs (LSA type 5). There should be one for 10/8, and one for 192.168.3.3/23.
- Notice that on R2, the LSAs are marked with * , indicating that these LSAs were created locally.
- Notice that subnet mask is not displayed. If you are curious enter `show ospf database extensive external` to see more information:

```
user@R2> show ospf database extensive external
    OSPF AS SCOPE link state database
 Type       ID               Adv Rtr           Seq      Age  Opt  Cksum  Len
Extern  *10.0.0.0         192.168.2.2      0x80000001   470  0x22 0x58e2  36
  mask 255.0.0.0
  Topology default (ID 0)
    Type: 2, Metric: 0, Fwd addr: 0.0.0.0, Tag: 0.0.0.0
  Gen timer 00:30:51
  Aging timer 00:52:10
  Installed 00:07:50 ago, expires in 00:52:10, sent 00:07:50 ago
  Last changed 00:07:50 ago, Change count: 1, Ours
Extern  *192.168.3.3      192.168.2.2      0x80000001   470  0x22 0xe9eb  36
  mask 255.255.255.255
  Topology default (ID 0)
    Type: 2, Metric: 0, Fwd addr: 0.0.0.0, Tag: 0.0.0.0
  Gen timer 00:42:10
  Aging timer 00:52:10
  Installed 00:07:50 ago, expires in 00:52:10, sent 00:07:50 ago
  Last changed 00:07:50 ago, Change count: 1, Ours
```

You can see the subnet mask, the metric, and the external LSA type (which is 2 by default).

Check the routing table on R1:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.0.0.0/8          *[OSPF/150] 00:12:47, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 12:12:14, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:12:47, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 12:13:04, metric 1
                       MultiRecv
```

We are checking for:

- Routes for 10.0.0.0/8 and 192.168.3.3/32, which should be learned from OSPF and have a preference value of 150 (default preference value for OSPF external routes – routes that were redistributed into OSPF).
- Correct next hop and interface.
- The routes are active.

Variations to the Scenario

What if we need to advertise the same static routes from R2 to R1 with a specific metric? In this case, a modifying action that changes the metric needs to be added to the policy on R2.

To change the policy:

```
[edit policy-options policy-statement redistribute-static]
user@R2# set term 1 then metric 100

[edit policy-options policy-statement redistribute-static]
user@R2# show | display set relative
set term 1 from protocol static
set term 1 from route-filter 10.0.0.0/8 exact
set term 1 from route-filter 192.168.3.3/32 exact
set term 1 then metric 100
set term 1 then accept

[edit policy-options policy-statement redistribute-static]
user@R2# commit
commit complete
```

Check the routing table on R1 again:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/8          *[OSPF/150] 00:00:35, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 12:50:43, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:00:35, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 12:51:33, metric 1
                       MultiRecv
```

What if you only wanted to modify the metric for the 10/8 route? There are actually many ways to achieve this. You could replace term 1 with two terms: one for 10/8 that changes the metric, and one term for 192.168.3.3/32 that does not change the metric, which would look like this:

```
[edit policy-options policy-statement redistribute-static]
user@R2# show | display set relative
set term 1 from route-filter 10.0.0.0/8 exact
set term 1 then metric 100
set term 1 then accept
set term 2 from route-filter 192.168.3.3/32 exact
set term 2 then accept
```

Or you could simply add the metric to the static route for 10/8 and the value would be exported automatically. Add the metric to the static route:

```
[edit routing-options static
user@R2# set route 10.0.0.0/8 metric 100

[edit policy-options policy-statement redistribute-static]
user@R2# commit
commit complete
```

Check the routing table on R1 again:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/8          *[OSPF/150] 00:00:07, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 13:05:12, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:11:47, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 13:06:02, metric 1
                       MultiRecv
```

How can we achieve the same result (redistribute the two static routes) but using a prefix-list? We need to define a prefix-list under the `edit policy-options` stanza using:

```
set prefix-list <prefix-list-name> <prefix>
```

Then we need to modify the policy and instead of route-filters reference the prefix-list:

```
set from prefix-list <prefix-list-name>
```

Create the prefix list:

```
[edit policy-options prefix-list STATICS]
user@R2# show | display set relative
set 10.0.0.0/8
set 192.168.3.3/32
```

```
[edit policy-options prefix-list STATICS]
user@R2# show
10.0.0.0/8;
192.168.3.3/32;
```

Modify the policy:

```
[edit policy-options policy-statement redistribute-static]
user@R2# delete term 1 from route-filter 10.0.0.0/8 exact

[edit policy-options policy-statement redistribute-static]
user@R2# delete term 1 from route-filter 192.168.3.3/32 exact

[edit policy-options policy-statement redistribute-static]
user@R1# set term 1 from prefix-list STATICS

[edit policy-options policy-statement redistribute-static]
user@R2# commit
commit complete
```

Check the routing table on R1 again to make sure the routes are there:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/8          *[OSPF/150] 00:21:23, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 13:26:28, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:33:03, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 13:27:18, metric 1
                       MultiRecv
```

NOTE The metric for 10/8 is 100, because remember we added it to the static route.

What if a new static route was added to R2 for 10.1.1/24? Would this route be advertised with our current configuration? Let's try! Add the new static route on R2:

```
[edit routing-options]
user@R2# set static route 10.1.1.0/24 next-hop 172.16.2.2

[edit routing-options]
user@R2# commit
commit complete
```

Check the routing table on R1 again:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/8          *[OSPF/150] 00:25:58, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 13:31:03, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:37:38, metric 0, tag 0
```

```
                    > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 13:31:53, metric 1
                      MultiRecv
```

Is the 10.1.1.0/24 route there? Nope!

Why? Because when we reference a prefix-list with the `from prefix-list` option, it is *always exact*. Thus, our current policy is matching on exactly 10/8 only.

Remember that:

- 10/8 exact = aggregate (entire range)
- 10/8 orlonger = 10/8 aggregate AND any subnet
- 10/8 longer = any subnet of 10/8

How do we fix it? Using *from prefix-list-filter* instead.

Modify the policy:

```
[edit policy-options policy-statement modified-redistribute-static]
user@R2# delete term 1 from prefix-list STATICS

[edit policy-options policy-statement modified-redistribute-static]
user@R2# set term 1 from prefix-list-filter STATICS orlonger

[edit policy-options policy-statement modified-redistribute-static]
user@R2# commit
commit complete
```

Check the routing table on R1 one more time:

```
user@R1> show route protocol ospf
inet.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/8          *[OSPF/150] 00:32:38, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
10.1.1.0/24         *[OSPF/150] 00:01:19, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.2.2/32      *[OSPF/10] 13:37:43, metric 1
                     > to 172.16.1.2 via fe-0/0/0.0
192.168.3.3/32      *[OSPF/150] 00:44:18, metric 0, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
224.0.0.5/32        *[OSPF/10] 13:38:33, metric 1
                      MultiRecv
```

Now the new route is there!!

Hopefully, these examples gave you a good grasp on how to implement policies to redistribute routes. Now let's switch gears and look at another use case of policies.

Load Balancing

Here is our new topology in Figure 6.31b.

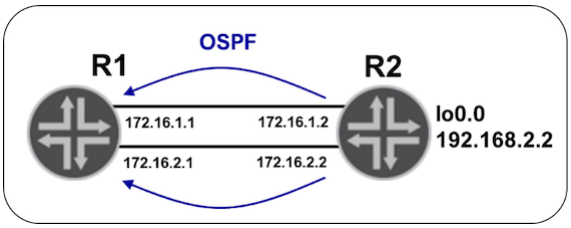


Figure 6.31b Load Balancing Example Topology

R1 and R2 are configured like this:

```
lab@R1# show | display set relative
set protocols ospf area 0.0.0.0 interface lo0.0
set protocols ospf area 0.0.0.0 interface ge-0/0/1.0
set protocols ospf area 0.0.0.0 interface ge-0/0/2.0

lab@R2# show | display set relative
set protocols ospf area 0.0.0.0 interface lo0.0
set protocols ospf area 0.0.0.0 interface ge-0/0/1.0
set protocols ospf area 0.0.0.0 interface ge-0/0/2.0
```

And advertising their loopback interfaces to each other. When we check the routing table on R1:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.2/32     *[OSPF/10] 00:03:03, metric 1
                      to 172.16.1.2 via fe-0/0/0.0
                    > to 172.16.2.2 via fe-0/0/2.0
224.0.0.5/32       *[OSPF/10] 13:58:46, metric 1
                      MultiRecv
```

You can see the route for 192.168.2.2/32 with two possible next-hops: 172.16.1.2 and 172.16.2.2. However, when you look at the forwarding table:

```
user@R1> show route forwarding-table destination 192.168.2.2/32
Routing table: default.inet
Internet:
Destination     Type RtRef Next hop      Type Index NhRef Netif
192.168.2.2/32  user     0 172.16.2.2    ucst   584     4 fe-0/0/2.0
```

You only see one next hop. Remember: load balancing is *not* enabled by default. We can use a very simple policy applied to the forwarding-table to enable it.

We configure the policy like any other policy under the `edit policy-options` stanza, and then we apply the policy under routing-options with:

```
set forwarding-table export <policy-name>
```

We know it is an export policy because we are taking routes *from* the routing table and placing them into the forwarding table. There is a lot more to learn about load balancing, but we are going to focus just on the policy part of it.

Create the policy:

```
[edit policy-options policy-statement load-balance]
user@R1# set term 1 then load-balance per-packet
```

We are calling the policy load-balance because it makes sense, but you can call it whatever you want. Notice that the action of the policy is `load-balance per-packet`. This is just enabling load balancing. The routers these days actually perform per-flow (per conversation) load balancing but the command was never changed.

Since we don't have a `from` statement, this action will be applied to all routes with two or more next hops in the routing table. If we wanted to, we could add one of more `from` statements and only implement load balancing for some destinations.

Apply the policy:

```
[edit routing-options]
user@R1# set forwarding-table export load-balance

[edit routing-options]
user@R1# commit
commit complete
```

Check the routing table again:

```
user@R1> show route protocol ospf
inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.2/32     *[OSPF/10] 00:22:39, metric 1
                      to 172.16.1.2 via fe-0/0/0.0
                    > to 172.16.2.2 via fe-0/0/2.0
224.0.0.5/32       *[OSPF/10] 14:18:22, metric 1
                      MultiRecv
```

No difference! But, let's check the forwarding table again:

```
user@R1> show route forwarding-table destination 192.168.2.2/32
Routing table: default.inet
Internet:
Destination     Type RtRef Next hop    Type Index  NhRef Netif
192.168.2.2/32 user     0   ulst 262142 2
               172.16.1.2 ucst 586      2 fe-0/0/0.0
               172.16.2.2 ucst 584      2 fe-0/0/2.0
```

Now we can see the two next ops!

Chapter 7

Troubleshooting Policies

By Yasmin Lara

You have seen many examples throughout Chapter Six that showed you how to configure routing policies and highlighted things to keep an eye on. This chapter contains some examples that can help you keep those details in mind when troubleshooting your policies in the future.

First, let's see if you can think of what could be wrong with the examples in the following list, before you look at the explanations directly following.

Missing terminating action:

```
[edit policy-options policy-statement POLICY-1]
user@R1# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 then metric 100
set term 1 then tag add 100

[edit protocols ospf]
user@R1# set export POLICY-1
```

Missing from statement:

```
[edit policy-options policy-statement POLICY-2]
user@R2# show| display set relative
set term 1 from route-filter 10.0.0.0/8 orlonger
set term 1 then accept

[edit protocols ospf]
user@R2# set export POLICY-2
```

Terms out of order:

```
[edit policy-options policy-statement POLICY-3]
user@R1# show | display set relative
```

```
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then accept
set term 2 from route-filter 10.1.1.128/26 exact
set term 2 then reject

[edit protocols ospf]
user@R1# set export POLICY-3
```

Policies out of order:

```
[edit policy-options policy-statement METRIC-100]
user@R1# show | display set relative
set from protocol static
set from route-filter 10.1.1.128/26 exact
set then metric 100
set then accept

[edit policy-options policy-statement METRIC-DEFAULT]
user@R1# show | display set relative
set from protocol static
set from route-filter 10.0.0.0/8 orlonger
set then accept

[edit protocols ospf]
user@R1# show | display set relative
set export METRIC-DEFAULT
set export METRIC-100
```

Wrong match type:

```
[edit policy-options policy-statement POLICY-4]
user@R1# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 longer
set term 1 then accept

[edit protocols ospf]
user@R1# show | display set relative
set export POLICY-4
```

Wrong direction:

```
[edit policy-options policy-statement POLICY-4]
user@R1# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 longer
set term 1 then accept

[edit protocols ospf]
user@R1# set import POLICY-4
```

Wrong protocol:

```
[edit policy-options policy-statement POLICY-5]
user@R1# show | display set relative
set term 1 from protocol bgp
set term 1 from route-filter 15.0.0.0/8 orlonger
set term 1 then accept
```

```
[edit protocols bgp]
user@R1# set export POLICY-5
```

Unnamed term:

```
[edit policy-options policy-statement POLICY-6]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set then accept

[edit protocols bgp]
user@R1# set export POLICY-6
```

Multiple route-filters configured incorrectly within a term (longest match goes wrong).

Figure 7.1 shows the topology for this chapter's troubleshooting examples.

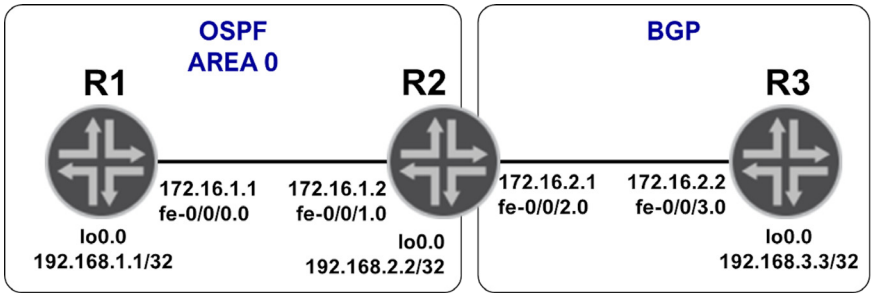


Figure 7.1 Topology Examples for Chapter 7

Missing Terminating Action

What you were trying to achieve

You want to redistribute a static route for 10.1.1/24 into OSPF.

Your configuration

```
[edit policy-options policy-statement POLICY-1]
user@R2# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 exact
set term 1 then metric 100
set term 1 then tag add 100

[edit protocols ospf]
user@R2# set export POLICY-1
```

Symptoms

External LSA not created by R2:

```
show ospf database | match extern | match 10\
```

…does not show anything.

Route is not present in the routing table on receiving side (R1):

```
show route protocol ospf terse | match 10\
```

…does not show anything.

What is the problem

Since there is *NO terminating action* in term 1 of policy POLICY-1, the route is processed against the default OSPF policy and is rejected. Remember the default export policy for OSPF is reject everything.

How to fix it

Add terminating action `accept` to the term:

```
[edit policy-options policy-statement POLICY-1]
user@R2# set term 1 then accept
```

Results

External LSA was created by R2:

```
user@R2> show ospf database | match extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000003    74  0x22 0x3836  36
```

The route is now present in the routing table on receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150        100            >172.16.1.2
```

Missing From Statement

What you were trying to achieve

You want to redistribute static routes for 10.1.1.0/24 and 10.2.2.0/24 into OSPF. You do not want to advertise other routes.

Your configuration

```
[edit policy-options policy-statement POLICY-2]
user@R2# show| display set relative
set term 1 from route-filter 10.0.0.0/8 orlonger
set term 1 then accept

[edit protocols ospf]
user@R2# set export POLICY-2
```

Symptoms

Some unexpected external LSAs are created by R2:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000002    61  0x22 0x3ff8  36
Extern  *10.1.2.0         192.168.2.2      0x80000001    61  0x22 0x3602  36
Extern  *10.2.2.0         192.168.2.2      0x80000001    61  0x22 0x2a0d  36
Extern  *10.2.2.16        192.168.2.2      0x80000001    61  0x22 0x2f07  36
Extern  *10.2.2.32        192.168.2.2      0x80000001    61  0x22 0x8e97  36
```

Some unexpected routes are present on the receiving side (R1):

```
user@R1> show route protocol ospf 10/8 terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.2.0/24        O 150          0            >172.16.1.2
* 10.2.2.16/28       O 150          0            >172.16.1.2
* 10.2.2.32/28       O 150          0            >172.16.1.2
```

What is the problem

R2 has routes matching 10/8 that it has learned from BGP:

```
user@R2> show route 10/8 terse | match 10\.
* 10.1.1.0/24          S    5                      >172.16.2.2
                       B 170        100           >172.16.2.2       200 I
* 10.1.2.0/24        B 170        100            >172.16.2.2       200 I
* 10.2.2.0/24          S    5                      >172.16.2.2
* 10.2.2.16/28       B 170        100            >172.16.2.2       200 I
* 10.2.2.32/28       B 170        100            >172.16.2.2       200 I
```

The policy on R2:

```
[edit policy-options policy-statement POLICY-2]
user@R2# show | display set relative
set term 1 from route-filter 10.0.0.0/8 orlonger
set term 1 then accept
```

…is matching on prefix 10/8 orlonger, without specifying the source of the routes. Thus, both the static and the BGP routes are matching, and being redistributed.

How to fix it

Add a from statement that specifies that only static routes should match:

```
[edit policy-options policy-statement POLICY-2]
user@R2# set term 1 protocol static
```

Results

Only the required external LSAs are created by R2:

```
user@R2> show ospf database | match extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000004    30  0x22 0x3bfa  36
Extern  *10.2.2.0         192.168.2.2      0x80000001    30  0x22 0x2a0d  36
```

Only the expected routes are present on the receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.2.2.0/24        O 150          0            >172.16.1.2
```

Terms Out of Order

What you were trying to achieve

You want to redistribute all subnets of 10.1/16 except for 10.1.1.128/26.

Your configuration

```
[edit policy-options policy-statement POLICY-3]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then accept
set term 2 from route-filter 10.1.1.128/26 exact
set term 2 then reject

[edit protocols ospf]
user@R1# set export POLICY-3
```

Symptom

R2 is creating an external LSA for 10.1.1.128/26 when it should not:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000004   807  0x22 0x3bfa  36
Extern  *10.1.1.63        192.168.2.2      0x80000001   242  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001   242  0x22 0xc037  36
```

The route for 10.1.1.128/26 is present on the receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.1.128/26      O 150          0            >172.16.1.2
```

What is the problem

When the route for 10.1.1.128/26 is evaluated against your policy, it matches the first term and gets accepted:

```
[edit policy-options policy-statement POLICY-3]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then accept
set term 2 from route-filter 10.1.1.128/26 exact
set term 2 then reject
```

How to fix it

The terms need to be swapped. The most specific terms should be placed before the least specific:

```
[edit policy-options policy-statement POLICY-3]
user@R1# insert term 2 before term 1

[edit policy-options policy-statement POLICY-3]
user@R1# show | display set relative
set term 2 from route-filter 10.1.1.128/26 exact
set term 2 then reject
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then accept
```

Results

R2 is no longer creating an external LSA for 10.1.1.128/26:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000005   305  0x22 0x39fb  36
Extern  *10.1.1.63        192.168.2.2      0x80000001  1303  0x22 0x4deb  36
```

The route for 10.1.1.128/26 is no longer present on the receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
```

Policies Out of Order

What you were trying to achieve

You want to redistribute all subnets of 10.1/16 with the default metric, except for 10.1.1.128/26, which you want to redistribute with a metric of 100.

Your configuration

```
[edit policy-options]
user@R2# show | display set relative | match METRIC
set policy-statement METRIC-100 from protocol static
set policy-statement METRIC-100 from route-filter 10.1.1.128/26 exact
set policy-statement METRIC-100 then metric 100
set policy-statement METRIC-100 then accept
set policy-statement METRIC-DEFAULT from protocol static
set policy-statement METRIC-DEFAULT from route-filter 10.0.0.0/8 orlonger
set policy-statement METRIC-DEFAULT then accept

[edit protocols ospf]
user@R1# show | display set relative
set export METRIC-DEFAULT
set export METRIC-100

[edit protocols ospf]
user@R1# show
export [ METRIC-DEFAULT METRIC-100 ];
```

Symptom

The route for 10.1.1.128/26 is present on R1 but has the default metric. It should have a metric of 100:

```
user@R1> show route 10.1.1.128/26
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.1.1.128/26      *[OSPF/150] 10:07:50, metric 0, tag 0
                    > to 172.16.1.2 via fe-0/0/0.0
```

What is the problem

The policies were applied in this order:

```
export [ METRIC-DEFAULT METRIC-100 ];
```

The `METRIC-DEFAULT` policy matches 10.1.1.0/24 orlonger, so the route for 10.1.1.128/26 matches this term, and is accepted without the metric being changed.

How to fix it

Reorder the policies so that the `METRIC-100` policy is evaluated first:

```
[edit protocols ospf]
user@R1# insert export METRIC-100 before METRIC-DEFAULT

[edit protocols ospf]
user@R1# show
export [ METRIC-100 METRIC-DEFAULT ];
```

Result

```
user@R1> show route 10.1.1.128/26
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, – = Last Active, * = Both

10.1.1.128/26       *[OSPF/150] 00:00:10, metric 100, tag 0
                     > to 172.16.1.2 via fe-0/0/0.0
```

The route is now advertised with the correct metric.

Wrong Match Type

What you were trying to achieve

You want R2 to advertise 10.1.1.0/24 and any subnet of 10.1.1.0/24.

Your configuration

```
[edit policy-options policy-statement POLICY-4]
user@R1# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 longer
set term 1 then accept

[edit protocols ospf]
user@R1# set export POLICY-4
```

Symptoms

R2 is *not* creating an external LSA for 10.1.1.0/24. It is creating LSAs only for subnets 10.1.1.0/26 and 10.1.1.128/26:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.63        192.168.2.2      0x80000001   128  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001   128  0x22 0xc037  36
```

The route for 10.1.1.0/24 is NOT present on the receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.1.128/26      O 150          0            >172.16.1.2
```

What is the problem

```
[edit policy-options policy-statement POLICY-4]
user@R1# show | display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 longer
set term 1 then accept
```

Your policy is matching on 10.1.1.0/24 `longer` instead of `orlonger`. The route for 10.1.1.0/24 matches the number of bits but the subnet mask is *not* longer that /26.

How to fix it

Change the match type from `longer` to `orlonger`. You have to delete the route filter with the match type `longer` first:

```
[edit policy-options policy-statement POLICY-4]
user@R1# delete term 1 from route-filter 10.1.1.0/24 longer
```

And then add the new route-filter:

```
[edit policy-options policy-statement POLICY-4]
user@R1# set term 1 from route-filter 10.1.1.0/24 orlonger
```

You can also do this:

```
[edit policy-options policy-statement POLICY-4 term 1 from]
user@R1# rename route-filter 10.1.1.0/24 longer to route-filter 10.1.1.0/24 orlonger
```

NOTES

The `replace pattern` CLI option does not work in this case.

Also, configuring...

```
[edit policy-options policy-statement POLICY-4]
user@R1# set term 1 from route-filter 10.1.1.0/24 orlonger
```

...without deleting the current route-filter will create a second route-filter under the term, and will not fix the problem:

```
[edit policy-options policy-statement POLICY-4]
ylara@R1# show
Jul 17 05:55:37
term 1 {
    from {
        protocol static;
        route-filter 10.1.1.0/24 orlonger;
        route-filter 10.1.1.0/24 longer;
    }
    then accept;
}
```

Results

R2 is now creating external LSA for 10.1.1.0/24, 10.1.1.0/26, and 10.1.1.128/26:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000001    28  0x22 0x41f7  36
Extern  *10.1.1.63        192.168.2.2      0x80000001   812  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001   812  0x22 0xc037  36
```

The route for 10.1.1.0/24 is now also present on the routing table on R1:

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.1.128/26      O 150          0            >172.16.1.2
```

Wrong Direction

What you were trying to achieve

You want R2 to advertise 10.1.1.0/24 and any subnet of 10.1.1.0/24.

Your configuration

```
[edit policy-options policy-statement POLICY-4]
user@R1# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 longer
set term 1 then accept

[edit protocols ospf]
user@R1# set import POLICY-4
```

Symptoms

External LSA not created by R2:

```
user@R2> show ospf database | match extern | match 10\.
```

. . . does not show anything.

Route is not present in the routing table on receiving side (R1):

```
user@R1> show route protocol ospf terse | match 10\.
```

... does not show anything.

What is the problem

You applied the policy as an import policy instead of an export policy. You want the router to take *static routes already in the routing table* and send them to OSPF neighbors (FROM STATIC => OSPF), thus it should be an export policy:

```
[edit protocols ospf]
user@R1# set import POLICY-4
```

How to fix it

```
Apply the policy as an export policy instead of an import policy. [edit protocols ospf]
user@R1# delete import

[edit protocols ospf]
user@R1# set export POLICY-4
```

NOTE

Technically, you don't have to delete the import policy, and you can just add set export POLICY-4 to achieve the desire behavior. However, set import POLICY-4 is wrong, and you should remove it, to avoid potential problems later on:

Results

R2 is now creating external LSA for 10.1.1.0/24, 10.1.1.0/26 and 10.1.1.128/26.

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.1.0         192.168.2.2      0x80000001     5  0x22 0x41f7  36
Extern  *10.1.1.63        192.168.2.2      0x80000001     5  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001     5  0x22 0xc037  36
```

The route for 10.1.1.0/24 is now also present on the routing table on R1:

```
user@R1> show route protocol ospf terse | match 10\.
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.1.128/26      O 150          0            >172.16.1.2
```

Wrong Protocol

What you were trying to achieve

R2 is receiving routes for subnets of 15.1.0.0/26 from BGP. You want to redistribute those routes into OSPF.

Your configuration

```
[edit policy-options policy-statement POLICY-5]
user@R1# show | display set relative
set term 1 from protocol bgp
set term 1 from route-filter 15.0.0.0/8 orlonger
set term 1 then accept

[edit protocols bgp]
user@R1# set export POLICY-5
```

Symptoms

R2 is receiving the routes but is *not* creating OSPF external LSAs for any of them:

```
user@R2> show route terse protocol bgp | match 15\.
* 15.1.0.0/24        B 170        100            >172.16.2.2       200 I
* 15.1.1.0/24        B 170        100            >172.16.2.2       200 I
* 15.1.2.0/24        B 170        100            >172.16.2.2       200 I
* 15.1.3.0/24        B 170        100            >172.16.2.2       200 I

user@R2> show ospf database | match Extern | match 15\.
(no output)
```

The routes are not present in the routing table on R1:

```
user@R1> show route protocol ospf terse | match 10\.
(no output)
```

What is the problem?

You applied the policy as an export policy to BGP instead of OSPF. You want the router to take BGP routes already in the routing table and send them to OSPF neighbors (FROM BGP => OSPF), thus is should be an export policy applied to OSPF:

```
[edit protocols bgp]
user@R1# set export POLICY-5
```

How to fix it

Apply the policy as an export policy to OSPF:

```
edit protocols ospf]
user@R1# set export POLICY-5

[edit protocols bgp]
user@R1# delete export POLICY-5
```

NOTE Technically, you don't have to delete the export policy from BGP, and you can just add set export POLICY-5 to OSPF to achieve the desired behavior. However, applying the policy to BGP is wrong, and you should remove it to avoid potential problems later on.

Results

R2 is now creating OSPF external LSAs for all expected prefixes:

```
user@R2> show ospf database | match Extern | match 15\.
Extern  *15.1.0.0         192.168.2.2      0x80000001    97  0x22 0xb2a   36
Extern  *15.1.1.0         192.168.2.2      0x80000001    97  0x22 0xff34  36
Extern  *15.1.2.0         192.168.2.2      0x80000001    97  0x22 0xf43e  36
Extern  *15.1.3.0         192.168.2.2      0x80000001    97  0x22 0xe948  36
```

The 15/8 subnet routes are now present in the routing table on R1:

```
user@R1> show route protocol ospf terse | match 15\.
* 15.1.0.0/24        O 150          0            >172.16.1.2
* 15.1.1.0/24        O 150          0            >172.16.1.2
* 15.1.2.0/24        O 150          0            >172.16.1.2
* 15.1.3.0/24        O 150          0            >172.16.1.2
```

Unnamed Term

What you were trying to achieve: you want R2 to redistribute the static route for 10.1.1.0/24 only?

Your configuration

```
[edit policy-options policy-statement POLICY-6]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set then accept

[edit protocols ospf]
user@R1# set export POLICY-6
```

Symptoms

R2 is creating OSPF external LSAs for all subnets of 10/8:

```
user@R2> show ospf database | match Extern | match 10\.
Extern  *10.1.0.0         192.168.2.2      0x80000001   134  0x22 0x4ced  36
Extern  *10.1.1.0         192.168.2.2      0x80000001   134  0x22 0x41f7  36
Extern  *10.1.1.63        192.168.2.2      0x80000001   134  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001   134  0x22 0xc037  36
Extern  *10.1.2.0         192.168.2.2      0x80000001   134  0x22 0x3602  36
Extern  *10.1.3.0         192.168.2.2      0x80000001   134  0x22 0x2b0c  36
Extern  *10.2.2.16        192.168.2.2      0x80000001   134  0x22 0x2f07  36
Extern  *10.2.2.32        192.168.2.2      0x80000001   134  0x22 0x8e97  36
```

In fact, it is creating external LSAs for *all* the routes in its routing table:

```
user@R2> show ospf database | match Extern
Extern  *10.1.0.0         192.168.2.2      0x80000001   170  0x22 0x4ced  36
Extern  *10.1.1.0         192.168.2.2      0x80000001   170  0x22 0x41f7  36
Extern  *10.1.1.63        192.168.2.2      0x80000001   170  0x22 0x4deb  36
Extern  *10.1.1.128       192.168.2.2      0x80000001   170  0x22 0xc037  36
Extern  *10.1.2.0         192.168.2.2      0x80000001   170  0x22 0x3602  36
Extern  *10.1.3.0         192.168.2.2      0x80000001   170  0x22 0x2b0c  36
Extern  *10.2.2.16        192.168.2.2      0x80000001   170  0x22 0x2f07  36
Extern  *10.2.2.32        192.168.2.2      0x80000001   170  0x22 0x8e97  36
Extern  *15.1.0.0         192.168.2.2      0x80000001   839  0x22 0xb2a   36
Extern  *15.1.1.0         192.168.2.2      0x80000001   839  0x22 0xff34  36
Extern  *15.1.2.0         192.168.2.2      0x80000001   839  0x22 0xf43e  36
Extern  *15.1.3.0         192.168.2.2      0x80000001   839  0x22 0xe948  36
Extern  *192.168.3.3      192.168.2.2      0x80000001   170  0x22 0xe9eb  36
```

There are unexpected OSPF routes in the routing table on R1 :

```
user@R1> show route protocol ospf terse
inet.0: 19 destinations, 19 routes (19 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.0.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/24        O 150          0            >172.16.1.2
* 10.1.1.0/26        O 150          0            >172.16.1.2
* 10.1.1.128/26      O 150          0            >172.16.1.2
* 10.1.2.0/24        O 150          0            >172.16.1.2
* 10.1.3.0/24        O 150          0            >172.16.1.2
* 10.2.2.16/28       O 150          0            >172.16.1.2
* 10.2.2.32/28       O 150          0            >172.16.1.2
* 15.1.0.0/24        O 150          0            >172.16.1.2
* 15.1.1.0/24        O 150          0            >172.16.1.2
* 15.1.2.0/24        O 150          0            >172.16.1.2
* 15.1.3.0/24        O 150          0            >172.16.1.2
* 172.16.2.0/24      O  10          2            >172.16.1.2
* 192.168.2.2/32     O  10          1            >172.16.1.2
* 192.168.3.3/32     O 150          0            >172.16.1.2
```

What is the problem

When you created your policy, instead of typing `set term 1 then accept`, you typed `set then accept` (omitting the term):

```
[edit policy-options policy-statement POLICY-6]
user@R1# show | display set relative
set term 1 from route-filter 10.1.1.0/24 exact
set then accept
```

Thus, term 1 has no terminating action, and an accept action is applied to all routes at the end of the policy.

As a result, the routes for 10.1.1/24 and 10.2.2/24 are evaluated against term 1; they match, but then they are accepted by the accept action outside the term. This works for these two routes! However, any other route in the routing table is also evaluated against term 1 and even though they don't match, they are also accepted by the accept action outside the term.

How to fix it

Remove the accept action outside of term 1 and add it to term 1:

```
[edit policy-options policy-statement POLICY-6]
user@R1# delete then accept

[edit policy-options policy-statement POLICY-6]
user@R1# set term 1 then accept
```

Results

R2 is only creating an external LSA for 10.1.1.0/24:

```
user@R2> show ospf database | match Extern
Extern  *10.1.1.0         192.168.2.2      0x80000002   342  0x22 0x3ff8  36
```

The only OSPF external route present in the routing table on R1 is 10.1.1.0/24:

```
user@R1> show route protocol ospf terse | match 150
* 10.1.1.0/24        O 150          0            >172.16.1.2
```

NOTE

In Junos, having a from or then statement outside of an explicit term is called an *unnamed term*. At the beginning of this chapter we mentioned that you can have zero or more terms within your policy. Well,, having zero terms is also known as unnamed term, and it means you are entering your from statements and then statements directly under the policy.

The potential problem is that it is possible to have both types of terms within the same policy, and you can run into situations like the one we are describing in this example.

It is pretty easy to create these kinds of issues by mistake, so you have to be very careful. You can see how bad it can be. In our lab this mistake caused 12 more routes to be advertised. Imagine what could happen if R2 had the entire internet routing table with all 800k or so routes on it!

Also, keep in mind that when you configure policies this way, the *unnamed term is always placed at the end of the policy*, regardless of where in the policy you type it. Here's an example:

```
[edit policy-options policy-statement EXAMPLE]
user@R1# set term example-1 then accept

[edit policy-options policy-statement EXAMPLE]
user@R1# set then reject

[edit policy-options policy-statement EXAMPLE]
user@R1# set term example-3 then accept

[edit policy-options policy-statement EXAMPLE]
user@R1# set term example-then reject

[edit policy-options policy-statement EXAMPLE]
user@R1# show | display set relative
set term example-1  then accept
set term example-3  then accept
set term example-4  then reject
set then reject
```

With this simple policy, it's quite easy to visualize that the reject action is outside the term, (maybe your intention was to add the reject action under a term called example-2). You can also easily see how this action is moved to the bottom of the policy. In a longer, more complex policy, with possibly hundreds of terms, it might be harder to figure it out. You might waste time going through your terms trying to figure out which one is the one that is accepting/rejecting the routes that are not supposed to be accepted/rejected.

Multiple Route-Filters Configured Incorrectly Within a Term (Longest Match Goes Wrong)

What you were trying to achieve

You want static routes for any subnet of 10.1.1.0/24, except for 10.1.1.128/26 to be rejected. This should work even if new static routes are added to R2. Static routes for *any other prefix* should also be redistributed.

Your configuration

```
[edit policy-options policy-statement POLICY-7]
user@R2# show| display set relative
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 from route-filter 10.1.1.128/26 exact accept
set term 1 then reject
set term 2 from protocol static
set term 2 then accept

[edit protocols ospf]
user@R2# set export POLICY-7
```

Symptoms

R2 is creating external LSA for a subnet of 10.1.1/24 that you did not expect:

```
user@R2> show ospf database | match Extern | match 10.1.1
Extern  *10.1.1.128       192.168.2.2      0x80000001    62  0x22 0xc037  36
Extern  *10.1.1.160       192.168.2.2      0x80000001   193  0x22 0xa007  36
```

The route for this additional subnet is present in the routing table on R1:

```
user@R1> show route protocol ospf terse 10.1.1/24
inet.0: 12 destinations, 12 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.128/26      O 150          0            >172.16.1.2
* 10.1.1.160/28      O 150          0            >172.16.1.2
```

What is the problem

The answer is longest match! Let's go over the route-filter evaluation steps one more time:

Step1: Check that the value of the N bits in the route match the value of the N bits in the prefix defined in the filter (where N is the subnet mask)

First you pretend that the match type does not exist:

```
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 from route-filter 10.1.1.128/26 exact accept
```

Then find the longest match. We are evaluating the route for 10.1.1.160/28 to figure out why it is being accepted as seen in Table 7.1.

Table 7.1 Evaluating the Route

route-filter	Meaning	Route under evaluation 10.1.1.160/28	Does it match?
10.1.1.0/24	The first 24 bits must match 10.1.1	10.1.1.	YES
10.1.1.128/26	The first 26 bits must match 10.1.1.10000000	10.1.1.10100000	YES

The route 10.1.1.160/28 matches both route filters, but the longest match is the second one. The router moves on to step 2 for this route filter (and only for this route filter).

Step 2: Compare the number of bits in route's subnet mask with the number of bits in the filter's subnet mask according to the subnet mask match type configured in the filter:

```
set term 1 from route-filter 10.1.1.128/26 exact accept
```

The route filter specifies a subnet mask of exactly 26 bits and the route has a subnet mask of /28. Thus, 10.1.1.160/28 does *not* match the subnet mask part of the route filter *and* therefore does *not* match the term.

Remember that once the longest match has been found, the router does *not* go back to the term to check any other route filter! As a result, the route for 10.1.1.160/28 is evaluated against term 2, which matches on all static routes, and does not specify any route-filter. Thus the route is accepted.

How to fix it

There are different ways to fix this but probably the easiest is to separate the `route-filters` into two different terms :

```
[edit policy-options policy-statement POLICY-7]
user@R1# delete term 1

[edit policy-options policy-statement POLICY-7]
user@R1# set term 0 from protocol static

[edit policy-options policy-statement POLICY-7]
user@R1# set term 0 from route-filter 10.1.1.128/26 exact

[edit policy-options policy-statement POLICY-7]
user@R1# set term 0 then accept

[edit policy-options policy-statement POLICY-7]
user@R1# set term 1 from protocol static

[edit policy-options policy-statement POLICY-7]
user@R1# set term 1 from route-filter 10.1.1/24 orlonger

[edit policy-options policy-statement POLICY-7]
user@R1# set term 1 then reject

[edit policy-options policy-statement POLICY-7]
user@R1# show | display set relative
set term 2 from protocol static
set term 2 then accept
set term 0 from protocol static
set term 0 from route-filter 10.1.1.128/26 exact
set term 0 then accept
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then reject

[edit policy-options policy-statement POLICY-7]
user@R1# insert term 2 after term 1

[edit policy-options policy-statement POLICY-7]
user@R1# show | display set relative
set term 0 from protocol static
set term 0 from route-filter 10.1.1.128/26 exact
set term 0 then accept
set term 1 from protocol static
set term 1 from route-filter 10.1.1.0/24 orlonger
set term 1 then reject
set term 2 from protocol static
set term 2 then accept
```

Now the route for 10.1.1.160/28 will not match term 0, it will be evaluated against term 1, it will match, and will be rejected.

Results

R2 is now creating external LSA only for subnet 10.1.1.128/26:

```
user@R2> show ospf database | match Extern | match 10.1.1
Extern  *10.1.1.128       192.168.2.2      0x80000001  1720  0x22 0xc037  36
```

Now the only subnet of 10.1.1/24 present in the routing table on R1 is 10.1.1.128/26:

```
user@R1> show route protocol ospf terse 10.1.1/24
inet.0: 11 destinations, 11 routes (11 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

A Destination        P Prf   Metric 1   Metric 2  Next hop         AS path
* 10.1.1.128/26      O 150          0            >172.16.1.2
```

Chapter 8

Firewall Filter Concepts

By Peter Klimai

Generally, a router will happily process and forward all transit packets for which it has a route to the packet's destination. Often, however, there are some special requirements for processing a subset of traffic. You may want to drop, rate-limit, prioritize, log, and so on – some but not all of the packets. Firewall filters allow you to solve this task by granularly controlling what is entering and what is leaving your network device.

In particular, among other things, you can do the following with firewall filters:

- Match packets using multiple fields in a packet header, such as IP protocol, source and/or destination address, port, and so on.
- Selectively drop or permit matched traffic.
- Count and log traffic, assign it to a forwarding class and change loss priority.
- Rate limit (police) traffic.
- Perform policy-based routing (PBR), also known as filter-based forwarding (FBF).
- Filter control plane traffic, that is, packets destined to your device's RE.

Figure 8.1 illustrates the firewall filter concept (figuratively). Indeed, traffic filtering is somewhat similar to air filtering, but remember that dropping some packets is only one of the use cases for firewall filters.

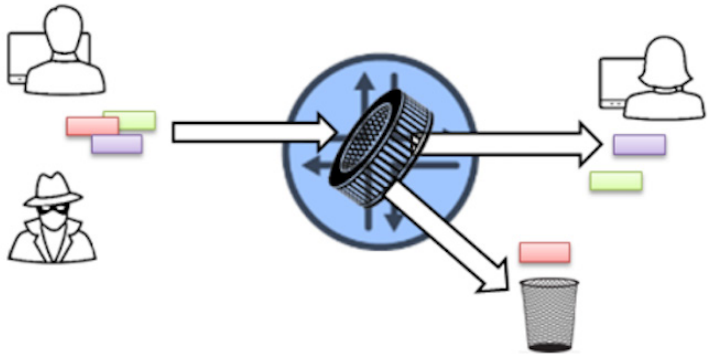


Figure 8.1 *Traffic Filtering Illustrated with Analogy*

Firewall filter functionality is common for all Junos-based devices. There can be some platform-specific details, for example, some but not all devices will support Layer 2 (family ethernet-switching) filters. This chapter focuses on Layer 3 IP (family inet) filters, but the main concepts for filtering other families' packets remain the same.

Note also that firewall filters are similar to access control lists (ACLs) in other vendor's terminology. Juniper's implementation of firewall filters uses highly efficient PFE-based processing and even very large filters have negligible performance impact.

An important thing to keep in mind is that firewall filters are stateless. This means that every packet is processed by a filter independently, and no information about previous packets is kept. In particular, there are no flow sessions and return traffic is not allowed automatically. This is opposed to stateful filtering that is supported on some of the Junos-based devices (most notably SRX Series).

As you will see from the rest of this chapter, firewall filters look rather similar to routing policies. Please make no mistake here: these are two significantly different entities. Just always keep in mind that routing policies work with `routes` and operate at the `control plane`, while firewall filters work with `packets` and operate at the `data plane`. That is, filters are enforced at the PFE rather than the RE.

Firewall Filter Structure

The basic filter structure is illustrated in Figure 8.2. Each filter has a name, in this example *my-filter*. A filter contains one or more terms, with each term having its own name and including `from` and `then` statements. From is basically a condition and then includes actions (and action modifiers) to take if the condition holds for the packet that is processed by a filter.

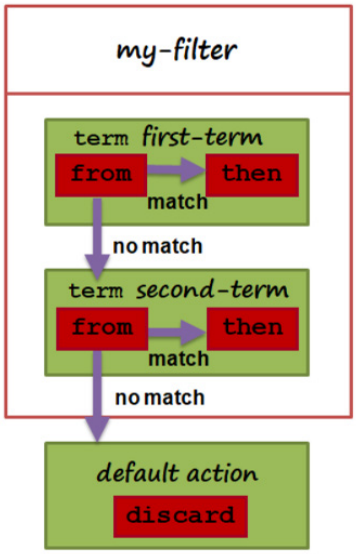


Figure 8.2 Firewall Filter Basic Structure

Below are some details to keep in mind related to Figure 8.2:

- Junos evaluates the filter's terms in sequence until it reaches a matching term with a terminating action. Terminating actions include accept, reject, and discard, as explained next.
- The from section can contain match criteria based on most fields in the packet header, such as port, IP protocol, source or destination address, and so on.
- The then section can contain one or more actions, including terminating actions (accept, reject, discard) and action modifiers (such as log, count, etc.)
- At the end of each filter, there is an implicit discard action. So all packets that are not explicitly allowed (accepted) are dropped.

Firewall Filter Action Details

In the filter's then block, you can specify terminating actions, action modifiers, and flow control actions. The following are three main `terminating actions` that can be used in a filter's term:

- `Accept` – Accept packet for forwarding.
- `Discard` – Silently drop the packet.

- `Reject` – Drop the packet but in a polite manner, that is, send a message to the source indicating that a packet was dropped. The message type is by default ICMP destination unreachable, but other message types can be configured instead, including TCP reset for TCP traffic.

Remember that having a terminating action in a term basically tells a Junos device to stop processing the following terms. The `accept`, `reject`, and `discard` are most common but not the only possible terminating actions – for example, `routing-instance` is another terminating action that directs packets to a specified routing instance, performing filter-based forwarding (FBF).

The question of what is the difference between `discard` and `reject`, and when to use each of them, may arise. Generally, use `reject` if you want to notify the user or application that they are being blocked. In this case the user will not have to wait until the application times-out which may be frustrating. But if you are using a filter to block an attack, use `discard` instead.

The following are some of the action modifiers that you can use in a filter:

- `count` – Count packets and bytes matching the term, using named counter.
- `log` – Log the packet in to a memory buffer (use `show firewall log` command to see the log).
- `syslog` – Log the packet using Junos syslog functionality (use `show log <filename>` command to see the log).
- `policer` – Apply policer to rate limit the traffic.

One important rule to keep in mind is this: if you have an action modifier in a term, a terminating action of `accept` is implied. So if you want to count and accept some traffic, you have two options that will lead to the same result:

- Using `count` in the `then` section of a filter, or
- Using both `count` and `accept`.

On the other hand, if you want to count and reject, just include `count` and `reject`.

But what if you want to count and proceed the filter evaluation to the next term? There is one special flow control action that you can use in a filter, together with action modifiers (such as `log` or `count`): `next term`. As the name indicates, it tells Junos to proceed to the next term rather than terminate at the current one. This is useful to override the default `accept` action that comes together with any action modifier.

Firewall Filter Configuration

For the following examples, the lab topology shown in Figure 8.3 will be used. Interfaces and routing are already configured, so generally there is IP reachability between host-A and host-B routing instances that emulate end hosts in the topology.

Local tunnel (lt) interfaces are used in the topology to connect routing instances, emulating physical Ethernet interfaces, so we can treat them the same way – in particular, apply some firewall filters.

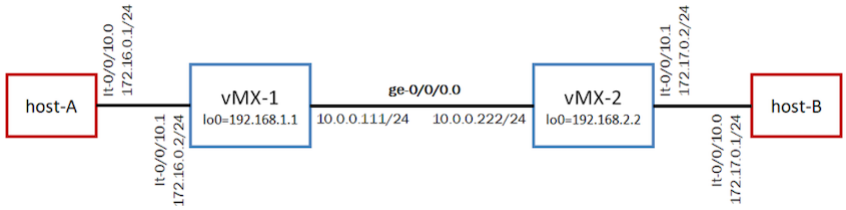


Figure 8.3 Chapter 8 Topology

The firewall filters for IP (family inet) traffic are configured in the `[edit firewall family inet]` hierarchy of a Junos configuration. Let's configure a very basic firewall filter that will allow only ICMP and HTTP traffic (using standard TCP port 80) on the vMX-1 device. Let's also log and count the matching traffic:

```
[edit]
lab@vMX-1# edit firewall family inet filter my-filter

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-icmp from protocol icmp

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-icmp then accept

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-icmp then log count icmp-counter

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-http from protocol tcp port http

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-http then accept

[edit firewall family inet filter my-filter]
lab@vMX-1# set term allow-http then log count http-counter

[edit firewall family inet filter my-filter]
lab@vMX-1# show
term allow-icmp {
    from {
```

```
        protocol icmp;
    }
    then {
        count icmp-counter;
        log;
        accept;
    }
}
term allow-http {
    from {
        protocol tcp;
        port http;
    }
    then {
        count http-counter;
        log;
        accept;
    }
}
```

Note a few things in regard to this configuration:

- Junos knows standard port and protocol names, so instead of `port 80` you can write `port http` and so on.
- Although the `accept` action is implied and can be omitted from the above configuration, it is a recommended best practice to explicitly specify it just for better clarity of your configuration.
- The `icmp-counter` and `http-counter` are user-defined counter names. You could use any other allowed name instead.
- All other packets except HTTP and ICMP will be silently dropped due to default `discard` term.

Applying Filter to an Interface

The filter that you just created is good, but by itself will do nothing before you apply it to an interface. Firewall filters are applied in the `input` or `output` direction to Junos device's logical interfaces.

NOTE Firewall filters can also sometimes be applied to other entities, such as a VLAN, or a router's forwarding table. These scenarios are beyond the scope of this book.

Looking at Figure 8.3, you can apply my-filter to lt-0/0/10.1 interface as input so it processes all traffic generated by host-A:

```
[edit firewall family inet filter my-filter]
lab@vMX-1# top

[edit]
lab@vMX-1# set interfaces lt-0/0/10 unit 1 family inet filter input my-filter
```

```
[edit]
lab@vMX-1# commit and-quit
commit complete
Exiting configuration mode
```

Now let's test our filter from vMX-1's host-A instance with some ping (ICMP) traffic:

```
lab@vMX-1> ping 172.17.0.1 routing-instance host-A count 10 rapid
PING 172.17.0.1 (172.17.0.1): 56 data bytes
!!!!!!!!!!
--- 172.17.0.1 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.876/1.362/2.190/0.392 ms

lab@vMX-1> show firewall

Filter: my-filter
Counters:
Name                                            Bytes           Packets
http-counter                                        0                 0
icmp-counter                                      840                10

Filter: __default_bpdu_filter__

lab@vMX-1> show firewall log
Log :
Time     Filter   Action Interface    Protocol       Src Addr                   Dest Addr
12:57:01  pfe      A     lt-0/0/10.1   ICMP          172.16.0.1                 172.17.0.1
12:57:01  pfe      A     lt-0/0/10.1   ICMP          172.16.0.1                 172.17.0.1
[…]
```

You can see that the ping traffic passes and it is reflected both in the firewall log and firewall counters.

Now to test the same with HTTP traffic, let's just initiate a fake telnet session to TCP port 80:

```
lab@vMX-1> telnet 172.17.0.1 port 80 routing-instance host-A
Trying 172.17.0.1...
telnet: connect to address 172.17.0.1: Connection refused
telnet: Unable to connect to remote host
lab@vMX-1> show firewall

Filter: my-filter
Counters:
Name                                            Bytes           Packets
http-counter                                       64                 1
icmp-counter                                      840                10

Filter: __default_bpdu_filter__
```

Although the HTTP connection is refused by host-B (it does not host an HTTP server), the packets on TCP port 80 are actually passing through, as you can see from the `http-counter` field.

What happens to other packets, for example those using a TCP port other than 80? Let's test for an example with port 8080:

```
lab@vMX-1> telnet 172.17.0.1 port 8080 routing-instance host-A
Trying 172.17.0.1...
telnet: connect to address 172.17.0.1: Operation timed out
telnet: Unable to connect to remote host
```

The operation times out after waiting for about 30 seconds, indicating that the packets were silently dropped. Remember, they are dropped by a default `discard` term that is present at the end of each filter.

If you wanted to allow all other traffic in addition to ICMP and HTTP, then an extra `permit all` term would be required, which you can configure as follows:

```
[edit firewall family inet filter my-filter]
lab@vMX-1# set term permit-all then accept
```

Alternatively, if you are OK with dropping all other traffic but want to monitor what is actually being dropped, you can instead do it this way:

```
[edit firewall family inet filter my-filter]
lab@vMX-1# set term discard-all then discard

[edit firewall family inet filter my-filter]
lab@vMX-1# set term discard-all then log count discard-all
```

As always, do not forget to commit your configuration to apply the change.

Firewall Filter Chains

On EX, MX, M, and T-series Junos-based devices, it is possible to apply multiple filters in a list, in the input or output direction. In this case, all terms of all filters are evaluated sequentially as they are configured. As an example, the filter permitting ICMP and HTTP traffic from a previous section is rewritten here as two filters:

```
[edit]
lab@vMX-1# show firewall
family inet {
   filter filter-icmp {
      term allow-icmp {
         from {
            protocol icmp;
         }
         then {
            count icmp-counter;
            log;
            accept;
         }
      }
   }
   filter filter-http {
```

```
        term allow-http {
            from {
                protocol tcp;
                port http;
            }
            then {
                count http-counter;
                log;
                accept;
            }
        }
    }
}

[edit]
lab@vMX-1# show interfaces lt-0/0/10 unit 1
encapsulation ethernet;
peer-unit 0;
family inet {
    filter {
        input-list [ filter-icmp filter-http ];
    }
    address 172.16.0.2/24;
}
```

Note how the two filters are applied, as a list, using the `input-list` statement under the interface. The result of filter list application in this example is exactly as before. Filter lists allow you to modularize your firewall filter configuration, similar to routing policy chains.

NOTE Another option for reusing firewall filters in multiple places is to reference a firewall filter from within the term of another stateless firewall filter. This is beyond the scope of this book.

Using Prefix Lists

You can use prefix lists to match packets' source and/or destination addresses in filters. Here is an example of a filter that only allows IP traffic to 192.168.2.0/24 and 172.17.0.0/24 subnets. All other traffic is dropped politely (using `reject`):

```
[edit]
lab@vMX-1# show firewall
family inet {
    filter only-my-subnets {
        term my-subnets {
            from {
                destination-prefix-list {
                    allowed-addresses;
                }
            }
            then accept;
        }
        term reject-else {
            then {
```

```
            reject;
         }
      }
   }
}

[edit]
lab@vMX-1# show policy-options
prefix-list allowed-addresses {
   172.17.0.0/24;
   192.168.2.0/24;
}

[edit]
lab@vMX-1# show interfaces lt-0/0/10 unit 1
encapsulation ethernet;
peer-unit 0;
family inet {
   filter {
      input only-my-subnets;
   }
   address 172.16.0.2/24;
}
```

Note that the `prefix-lists` that you use in firewall filters are the same prefix lists that you used with routing policies, so they are configured under the `[edit policy-options]` hierarchy.

And now let's perform a quick test:

```
lab@vMX-1> ping 172.17.0.1 routing-instance host-A rapid count 5
PING 172.17.0.1 (172.17.0.1): 56 data bytes
!!!!!
--- 172.17.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.134/1.533/2.009/0.295 ms

lab@vMX-1> ping 192.168.2.2 routing-instance host-A rapid count 5
PING 192.168.2.2 (192.168.2.2): 56 data bytes
!!!!!
--- 192.168.2.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.957/2.139/4.453/1.229 ms

lab@vMX-1> ping 10.0.0.222 routing-instance host-A rapid count 5
PING 10.0.0.222 (10.0.0.222): 56 data bytes
36 bytes from 172.16.0.2: Communication prohibited by filter
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
 4  5  00 0054 a993   0 0000  40  01 1a27 172.16.0.1  10.0.0.222
[…]
```

Notice that only addresses in the allowed subnets are *pingable*. Trying to ping vMX-2's interface address, 10.0.0.222, results in getting an ICMP error message in response, indicating that communication is prohibited by a filter.

Changing the Order of Terms

As stated previously, filter terms are evaluated from top to bottom, so the order of terms in a filter is important. Let's say you have a filter like this:

```
[edit]
lab@vMX-1# show firewall family inet filter allow-my-services
term http {
    from {
        protocol tcp;
        port http;
    }
    then accept;
}
term reject-else {
    then {
        reject;
    }
}
```

And you want to add a term allowing all GRE traffic. To do this, configure a term:

```
[edit]
lab@vMX-1# edit firewall family inet filter allow-my-services

[edit firewall family inet filter allow-my-services]
lab@vMX-1# set term gre from protocol gre

[edit firewall family inet filter allow-my-services]
lab@vMX-1# set term gre then accept

[edit firewall family inet filter allow-my-services]
lab@vMX-1# show
term http {
    from {
        protocol tcp;
        port http;
    }
    then accept;
}
term reject-else {
    then {
        reject;
    }
}
term gre {
    from {
        protocol gre;
    }
    then accept;
}
```

If you think about it, this will not work, as all GRE traffic will always match the term `reject-else`. So, to make the term `gre` effective, you need to move it before `reject-else`. Use the `insert` command to do it, this way:

```
[edit firewall family inet filter allow-my-services]
lab@vMX-1# insert term gre before term reject-else
```

```
[edit firewall family inet filter allow-my-services]
lab@vMX-1# show
term http {
    from {
        protocol tcp;
        port http;
    }
    then accept;
}
term gre {
    from {
        protocol gre;
    }
    then accept;
}
term reject-else {
    then {
        reject;
    }
}
```

Now, after committing the configuration, the order of terms is correct and the filter will work as expected (allowing both HTTP and GRE and rejecting everything else).

Using Policers

Policers are basically rate-limiters that allow you to constrain the speed for the given traffic. They can be used in conjunction with firewall filters, or separately (in this case they are just applied to a logical interface directly).

To start, let's look at an example policer configuration, named `limit-64K`:

```
[edit]
lab@vMX-1# show firewall policer limit-64K
if-exceeding {
    bandwidth-limit 64k;
    burst-size-limit 15k;
}
then discard;
```

Here, in the `if-exceeding` section, you specify the conditions for traffic after which an action configured in `then` will be applied. In this example, the excessive traffic is just discarded (hard policer), however, policers can also be used to change forwarding class and loss priority of traffic (soft policer).

Policers implement a token bucket algorithm that is illustrated in Figure 8.4.

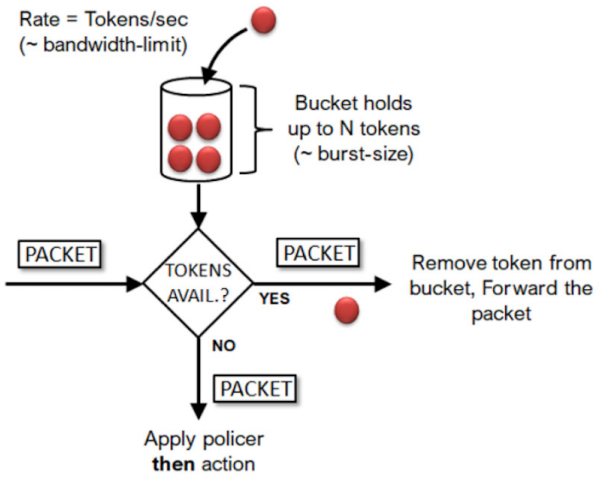


Figure 8.4 *Token Bucket Algorithm Illustration*

A few more notes on the algorithm:

- When you configure a policer, a router immediately imagines a virtual bucket and starts putting virtual tokens in it.
- Tokens are added to the bucket at a speed proportional to the `bandwidth-limit` that you configured.
- The size of the bucket is basically `burst-size` that you also configure in policer settings (for simplicity we omit *proportionality coefficients* in this explanation). If the bucket is already full, no new tokens are added.
- When a packet is evaluated against a policer, the router extracts a token (or a number of tokens proportional to the packet's size) from the bucket and forwards the packet as normal. This assumes that there are enough tokens in the bucket. If there are not enough tokens, policer action is applied to the packet. In the example above, the action is `discard`, so in this case the packet is just dropped.
- If there is no or little traffic for some time, the bucket eventually gets full, so a burst of packets of size `burst-size` can pass through the policer, exceeding the `bandwidth-limit` speed for some short interval of time.
- However, on the average, the algorithm guarantees that the average traffic speed does not exceed the `bandwidth-limit` (just because tokens are added at a limited speed, determined by `bandwidth-limit`).

- `Burst-size-limit` is measured and configured in `bytes` (kilobytes, megabytes, etc.). However, `bandwidth-limit` is configured in `bits per second` (or kilobits per second, etc.).
- Standard recommendation for the `burst-size-limit` is to configure it to be about 5 milliseconds times the interface speed, but not less than 10 MTU values for the physical interface. For example, for Gigabit Ethernet interface we get (1/8)*0.005s*1,000,000,000b/s = 625,000 Bytes.

After you configure a policer, you can either apply it directly to the logical interface (in this case it will apply to all traffic entering or leaving the interface, depending on the direction of application), or you can use it in a firewall filter. In the next example, use a filter to apply the `limit-64K` policer to all traffic originating in the 172.16.0.0/24 subnet. Don't forget to apply the firewall filter to the interface, in its turn!

```
[edit]
lab@vMX-1# show firewall family inet filter use-policer
term limit-local-subnet {
   from {
      source-address {
         172.16.0.0/24;
      }
   }
   then policer limit-64K;
}
term else-accept {
   then accept;
}

[edit]
lab@vMX-1# show interfaces lt-0/0/10 unit 1
encapsulation ethernet;
peer-unit 0;
family inet {
   filter {
      input use-policer;
   }
   address 172.16.0.2/24;
}
```

Now let's initiate some ping traffic. This time you do it in a bit of a tricky manner: rapid ping using 1000-Byte packets. This causes the potential interface load to be somewhat higher than 64K:

```
lab@vMX-1> ping 192.168.2.2 rapid count 100 size 1000 routing-instance host-A
PING 192.168.2.2 (192.168.2.2): 1000 data bytes
!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!!.!!!.!!!!.!!!!.!!!!!!!.!!!
--- 192.168.2.2 ping statistics ---
100 packets transmitted, 81 packets received, 19% packet loss
round-trip min/avg/max/stddev = 0.749/6.817/107.649/19.623 ms
```

As you see, some packets pass (!) and some are dropped (.). Looking at the interface statistics (by opening a different terminal session while ping command is running), you can confirm that traffic speed on an interface is rather close to what is configured in a policer:

```
lab@vMX-1> show interfaces lt-0/0/10.1 detail | find Transit statistics
   Transit statistics:
    Input  bytes  :           1680780             67472 bps
    Output bytes  :            423692             55560 bps
    Input  packets:              1377                 8 pps
    Output packets:               482                 6 pps
   Protocol inet, MTU: 1500
   Max nh cache: 0, New hold nh limit: 0, Curr nh cnt: 0, Curr new hold cnt: 0, NH drop cnt: 0
   Generation: 177, Route table: 0
     Flags: Sendbcast-pkt-to-re
     Input Filters: use-policer
     Addresses, Flags: Is-Preferred Is-Primary
       Destination: 172.16.0.0/24, Local: 172.16.0.2, Broadcast: 172.16.0.255, Generation: 164
```

Filtering Control Plane Traffic

One very important application of firewall filters is protecting local host (control plane) traffic entering the network devices. This includes management sessions (SSH, FTP, Telnet, Web management, NETCONF, REST API, etc.), routing protocol sessions (BGP, OSPF, RIP, etc.) and auxiliary services that router uses or provides (RADIUS, DNS, NTP, etc.). Normally, such packets are processed by the router's RE (see Figure 8.5).

To protect your RE from possible attacks, you want to filter such traffic. To do this, you use a special firewall filter, applied to the Junos device's loopback interface (lo0).

NOTE Don't try to understand this. Just remember that control plane traffic filtering is accomplished by applying a firewall filter to the loopback interface. Loopback interface is not the same as the interface that connects the RE and PFE, but it is the interface to which you should apply that filter! Keep in mind that the lo0-based filter only affects management and control plane traffic, not transit traffic.

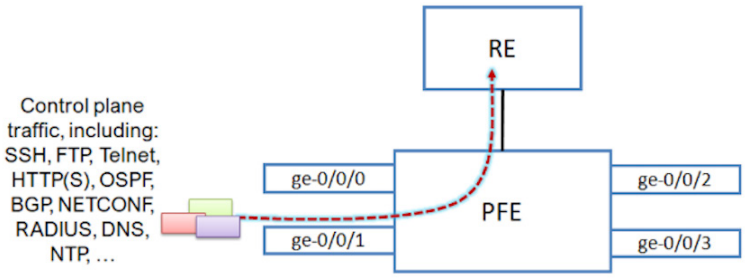


Figure 8.5 Traffic Going to the RE of a Junos Device

Before we look at an example loopback filter, note the following:

- Firewall filters are very sensitive to configuration error. This is even more so with loopback filters. Forgetting to include this or that protocol in the filter may easily lead to broken routing, broken network service, unhappy users, and to add to the problem, to you being dropped off a terminal session. So create filters very carefully.
- Even when creating filters carefully, there is still room for error. Junos `commit confirmed` command is your best friend when working with firewall filters, especially with filters applied to the lo0 interface.

There are different strategies that you can use to create a lo0 filter. One approach is to only permit all allowed protocols from corresponding allowed sources, and then drop everything else. In this case you need to really make sure you do not forget anything. Such a filter is typically going to be rather lengthy.

For an example, in this chapter let's focus on filtering only one type of management traffic – SSH. We will only allow SSH from the allowed list of subnets (defined in a `prefix-list`) and permit any other protocols from all sources at this stage. Here is a possible configuration of such a filter:

```
lab@vMX-1# show policy-options prefix-list ssh-allowed
10.254.0.0/24;
192.168.2.2/32;

[edit]
lab@vMX-1# show firewall family inet filter protect-re
term limit-ssh {
    from {
        source-address {
            0.0.0.0/0;
        }
        source-prefix-list {
            ssh-allowed except;
        }
        protocol tcp;
        port ssh;
    }
    then {
        log;
        discard;
    }
}
term else-accept {
    then accept;
}

[edit]
lab@vMX-1# show interfaces lo0
unit 0 {
    family inet {
        filter {
            input protect-re;
        }
        address 192.168.1.1/32;
    }
}
```

Note the following:

- The first filter's term is discarding SSH traffic from all hosts, `except` the allowed `prefix-list`.
- The second term allows every other packet, including SSH sessions, from the allowed `prefix-list` as well as all non-SSH traffic (in other words, everything that is not dropped by the first term).
- The filter is applied to the loopback interface, as it should be.

Now that the filter is applied to vMX-1's lo0.0 interface, let's try to SSH from vMX-2's lo0 address:

```
lab@vMX-2> ssh 192.168.1.1 source 192.168.2.2
Password: ******
Last login: Tue Apr  7 17:29:56 2020 from 10.254.0.251
--- JUNOS 18.3R1.9 Kernel 64-bit  JNPR-11.0-20180816.8630ec5_buil
lab@vMX-1> quit

Connection to 192.168.1.1 closed.

lab@vMX-2>
```

It works, because vMX-2's lo0 (192.168.2.2) is included in `prefix-list ssh-allowed`.

NOTE Although in this example we SSH to vMX-1's lo0 address, it is important to understand the lo0 filter will work regardless of which actual destination address on the router you use to initiate a session to.

Now let's SSH from vMX-2's physical interface address:

```
lab@vMX-2> ssh 192.168.1.1 source 10.0.0.222
ssh: connect to host 192.168.1.1 port 22: Operation timed out
As you see, SSH traffic is filtered properly. You could also check firewall log on vMX-1 to make sure
the drop is actually due to a filter:
[edit]
lab@vMX-1# run show firewall log
Log :
Time     Filter    Action Interface    Protocol       Src Addr                   Dest Addr
19:34:32  pfe      D     ge-0/0/0.0    TCP          10.0.0.222                 192.168.1.1
[…]
```

Now, a very important check. It turns out that vMX routers in this lab topology are running OSPF. It is important to make sure that our lo0 filter did not break OSPF (or any other control plane protocol – but here we limit ourselves to just checking OSPF):

```
lab@vMX-1# run show ospf neighbor
Address        Interface           State    ID            Pri  Dead
10.0.0.222      ge-0/0/0.0          Full     192.168.2.2     128   31
```

So far, looks good.

NOTE It's a good time to consider what would happen if you omitted the `else-accept` term from the `protect-re` filter. Would OSPF still work? (The answer is No!) What problems could you get in a real world situation with a similar filter?

Firewall Filters Default Behaviors

Working with firewall filters can be tricky because of several implicit filter behaviors that you need to keep in mind. Although this information was presented earlier, let's check that you got it all correctly once again.

Implicit Discard

Every filter has an implicit `discard` term at the end. This means that all traffic not explicitly allowed will be silently dropped.

So, for example, this filter:

```
[edit firewall family inet filter drop-something]
lab@vMX-1# show
term icmp {
    from {
        protocol icmp;
    }
    then {
        discard;
    }
}
```

…will drop everything: ICMP will be dropped by explicitly defined term while everything else will be discarded with a default term. Overlooking default discard term is a very common mistake.

No from Statement

If there is no `from` statement in the term, all packets will match it. So, to extend a previous example:

```
[edit firewall family inet filter drop-something]
lab@vMX-1# show
term icmp {
    from {
        protocol icmp;
    }
    then {
        discard;
    }
}
term else-accept {
    then accept;
}
```

In this case, all packets that are not ICMP will be allowed by the `else-accept` term.

Implicit Accept with Action Modifiers

Every action modifier such as `log` or `count` has an implicit `accept` action coming with it. So, the following term:

```
[edit firewall family inet filter ff]
lab@vMX-1# show
term allow-icmp {
    from {
        protocol icmp;
    }
    then log;
}
```

…is totally equivalent to this one:

```
[edit firewall family inet filter ff]
lab@vMX-1# show
term allow-icmp {
    from {
        protocol icmp;
    }
    then {
        log;
        accept;
    }
}
```

On the other hand, if you want to `log` and proceed to the next term, you can do it this way:

```
[edit firewall family inet filter ff]
lab@vMX-1# show
term allow-icmp {
    from {
        protocol icmp;
    }
    then {
        log;
        next term;
    }
}
```

No Action Specified

The corner case of a previous scenario is completely missing a `then` statement. In this case, the action that is implied is still `accept`. So the configuration can look like this:

```
[edit firewall family inet filter local-subnet]
lab@vMX-1# show
term local-subnet {
    from {
        source-address {
            172.16.0.0/24;
        }
    }
}
```

Although we highly discourage you from doing so, the above example is correct and will actually accept all traffic coming from the 172.16.0.0/24 subnet!

Specifics of Matching TCP Session Direction

Firewall filters allow you to match on multiple fields in the packet header. To get an idea of how rich this functionality is, try using context sensitive help (?), as follows:

```
lab@vMX-1# set firewall family inet filter ff term 1 from ?
Possible completions:
> address            Match IP source or destination address
+ apply-groups       Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
> destination-address  Match IP destination address
+ destination-class   Match destination class
+ destination-class-except  Do not match destination class
+ destination-port    Match TCP/UDP destination port
+ destination-port-except  Do not match TCP/UDP destination port
> destination-prefix-list  Match IP destination prefixes in named list
+ dscp              Match Differentiated Services (DiffServ) code point
+ dscp-except        Do not match Differentiated Services (DiffServ) code point
+ esp-spi           Match IPSec ESP SPI value
+ esp-spi-except      Do not match IPSec ESP SPI value
  first-fragment     Match if packet is the first fragment
> flexible-match-mask  Match flexible mask
> flexible-match-range  Match flexible range
+ forwarding-class    Match forwarding class
+ forwarding-class-except  Do not match forwarding class
  fragment-flags     Match fragment flags (in symbolic or hex formats) >>> (Ingress only)
+ fragment-offset     Match fragment offset
+ fragment-offset-except  Do not match fragment offset
+ gre-key           Match GRE Key
+ gre-key-except      Do not match GRE Key
+ icmp-code          Match ICMP message code
+ icmp-code-except    Do not match ICMP message code
+ icmp-type          Match ICMP message type
+ icmp-type-except    Do not match ICMP message type
> interface          Match interface name
+ interface-group     Match interface group
+ interface-group-except  Do not match interface group
> interface-set      Match interface in set
+ ip-options         Match IP options
+ ip-options-except   Do not match IP options
  is-fragment        Match if packet is a fragment
+ loss-priority      Match Loss Priority
+ loss-priority-except  Do not match Loss Priority
+ packet-length      Match packet length
+ packet-length-except  Do not match packet length
+ policy-map         Match policy map
+ policy-map-except   Do not match policy map
+ port              Match TCP/UDP source or destination port
+ port-except        Do not match TCP/UDP source or destination port
+ precedence         Match IP precedence value
+ precedence-except   Do not match IP precedence value
> prefix-list        Match IP source or destination prefixes in named list
+ protocol           Match IP protocol type
+ protocol-except     Do not match IP protocol type
+ rat-type           Match RAT Type
```

```
+ redirect-reason     Match Redirect Reason
  service-filter-hit  Match if service-filter-hit is set
> source-address     Match IP source address
+ source-class       Match source class
+ source-class-except  Do not match source class
+ source-port        Match TCP/UDP source port
+ source-port-except   Do not match TCP/UDP source port
> source-prefix-list   Match IP source prefixes in named list
  tcp-established     Match packet of an established TCP connection
  tcp-flags          Match TCP flags (in symbolic or hex formats)
  tcp-initial        Match initial packet of a TCP connection
+ ttl               Match IP ttl type
+ ttl-except         Do not match IP ttl type
```

Among other things, you can match on the TCP session state or specific TCP flags. For example, the `tcp-initial` option will only match the TCP SYN packet (first packet of the TCP session communication). This allows you to filter based on TCP traffic session initiation direction.

For example, let's create a firewall filter that will permit you to initiate TCP sessions (on any ports) from host-A to host-B, but not in the other direction (refer to Figure 8.3 for the topology). You can do it this way:

```
[edit]
lab@vMX-1# show firewall family inet filter tcp-in
term tcp-init {
   from {
      protocol tcp;
      tcp-initial;
   }
   then {
      reject;
   }
}
term else {
   then accept;
}

[edit]
lab@vMX-1# show interfaces lt-0/0/10 unit 1
encapsulation ethernet;
peer-unit 0;
family inet {
   filter {
      output tcp-in;
   }
   address 172.16.0.2/24;
}
```

Here you are using an output filter to reject all TCP SYN packets coming in the direction of host-A. This will not affect traffic of already established TCP sessions.

Let's check how it works. Initiate a telnet session from host-A to host-B:

```
[edit]
lab@vMX-1# run telnet 172.17.0.1 routing-instance host-A source 172.16.0.1
Trying 172.17.0.1...
Connected to 172.17.0.1.
```

```
Escape character is '^]'.
login: ^C
Connection closed by foreign host.
```

As you see, this works. In the other direction – from host-B to host-A:

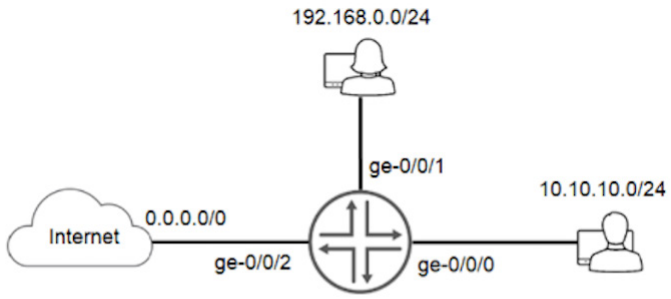
```
[edit]
lab@vMX-2# run telnet 172.16.0.1 routing-instance host-B source 172.17.0.1
Trying 172.16.0.1...
telnet: connect to address 172.16.0.1: Connection refused
telnet: Unable to connect to remote host
```

In this case, the TCP traffic is rejected because, again, the first packet of a TCP session is SYN, which matches the `tcp-initial` filtering criterion that you specified.

NOTE As you see, it is possible to do some filtering based on session initiation direction even with a `stateless` firewall filter. Still, be aware that `stateful` filtering (based on the flow or session table) would be more secure. For example, with firewall filter applied as in the above example, host-B could still send some bogus traffic (such as TCP ACK flood) and that would reach and potentially hurt host-A. A stateful firewall, such as Juniper SRX Series, would easily block such packets, but this is just an impossible task for a stateless filter.

Unicast Reverse-path-forwarding (uRPF)

Unicast reverse-path-forwarding (uRPF) is a method of protecting your network from IP address spoofing attack. To understand the attack and how you can protect from it, assume that you have a host in 10.10.10.0/24 subnet behind the ge-0/0/0 router interface (see Figure 8.6). The same router is connected to other networks, including the Internet.



Routing table:

Prefix	Interface
10.10.10.0/24	ge-0/0/0
192.168.0.0/24	ge-0/0/1
0.0.0.0/0	ge-0/0/2

Figure 8.6 Network Diagram for uRPF Illustration

It's easy to see that in such a topology packets with 10.10.10.0/24 source addresses must only enter the router via the ge-0/0/0 interface. If they enter via any other interface (such as ge-0/0/1 or ge-0/0/2), something is clearly wrong. Probably someone is trying to impersonate a private host to somehow avoid firewall filters applied on the router, or it may be a DoS attack.

NOTE One classical attack that uses IP spoofing is known as *LAND attack*. In the case of a LAND attack, an attacker will send TCP SYN packets with source addresses equal to the destination address of some server or host, trying to crash the system.

Anyhow, IP spoofing is generally bad and you want to block it in your networks. One way to avoid IP spoofing would be to use firewall filters. For example, you could easily check with a filter that packets entering ge-0/0/1 and ge-0/0/2 interfaces do not belong to the 10.10.10.0/24 subnet. However, this IP spoofing protection solution would be very unscalable, and there is a better way – uRPF.

The idea of uRPF is to detect IP spoofing by checking a reverse route for the packet.

NOTE Normally, routing happens by checking the packet's destination address against the routing table. So uRPF instead checks the source address. After (and if) uRPF checks are passed, the packet is routed as normal (using the destination address for route lookup).

uRPF works this way:

- When a packet arrives on the interface, the router performs a route lookup for the packet's source address. If you think about it, the result is basically a router interface beyond which the packet's source network is supposed to live.
- Compare the interface obtained from a reverse route lookup against the interface from which the packet was actually received.
- If the two interfaces match, everything is okay and there is no spoofing. If not, this packet should generally be treated as spoofed and discarded (see below to find out how Junos actually behaves: sometimes such packets will still be allowed).

Please feel free to re-read the previous paragraph because this concept is not trivial. Let's also take a look at two examples:

- Assume a packet with the source IP address of 10.10.10.10 is received on the ge-0/0/0 interface (Figure 8.6). Reverse route lookup operation returns via ge-0/0/0. Interfaces match so no spoofing is detected.
- Now assume a packet with the source IP address of 10.10.10.10 is received on the ge-0/0/1 interface. Reverse route lookup operation returns via ge-0/0/0. Interfaces do not match, so IP spoofing is detected (and such packets are typically going to be dropped – assuming uRPF is enabled, see next).

uRPF Modes in Junos

Junos allows you to enable uRPF on an interface, so all packets entering it will be checked using uRPF. There are two modes: *strict* and *loose*:

- Strict mode works this way: the packet is accepted only when the source address of the packet matches any of the routes that can be reachable through the interface. Routes can have multiple next hop interfaces associated with them; therefore, if one of the next hops matches the incoming interface of the packet, the packet is accepted.
- Loose mode works this way: the packet is accepted when it has a source address that matches any prefix in the routing table (regardless of the next hop, so uRPF does not actually compare incoming and RPF interfaces in this mode).

So, loose mode is basically just checking that there is some route for the packet's source in the routing table, and that's it. One gotcha with the loose mode is related to the default route:

- On all routers with MPCs and the MX80 router, for the purpose of loose mode uRPF check, the default route is treated as if the route does not exist.
- On all routers except those with MPCs and the MX80 router, all packets are automatically accepted by loose mode uRPF. For this reason, Juniper Networks *does not recommend configuring uRPF loose mode on interfaces that use default routes*.

Enabling uRPF in Junos

Let's first take a look at a simple IP spoofing attack. In our lab topology (Figure 8.7), host-B instance was configured with the 172.16.0.1 address as a loopback IP – note this address is the same as the 'legal' address of host-A instance.

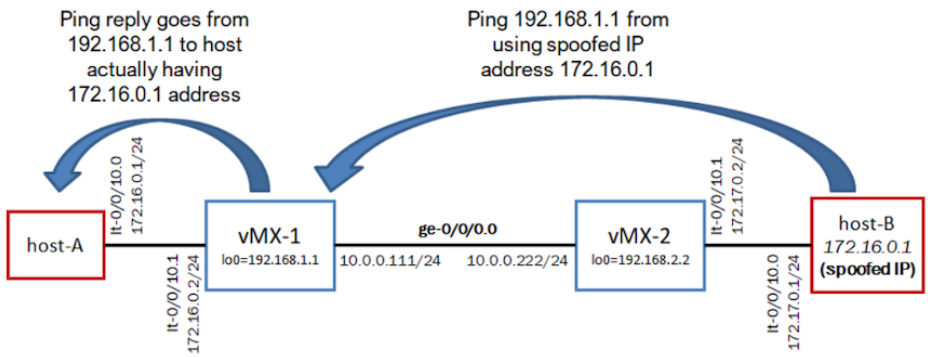


Figure 8.7 *IP Spoofing Lab Topology*

Now let's send a few spoofed packets (naturally, you do not get a response):

```
[edit]
lab@vMX-2# run ping 192.168.1.1 routing-instance host-B source 172.16.0.1 count 5
PING 192.168.1.1 (192.168.1.1): 56 data bytes

--- 192.168.1.1 ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
```

However, if you do monitor traffic on lt-0/0/10.0 interface of the vMX-1, this is what you will see:

```
lab@vMX-1# run monitor traffic interface lt-0/0/10.0
verbose output suppressed, use <detail> or <extensive> for full protocol decode
Address resolution is ON. Use <no-resolve> to avoid any reverse lookup delay.
Address resolution timeout is 4s.
Listening on lt-0/0/10.0, capture size 96 bytes

Reverse lookup for 192.168.1.1 failed (check DNS reachability).
Other reverse lookup failures will not be reported.
Use <no-resolve> to avoid reverse lookups on IP addresses.

10:46:01.926627  In IP 192.168.1.1 > 172.16.0.1: ICMP echo reply, id 55286, seq 0, length 64
10:46:02.927602  In IP 192.168.1.1 > 172.16.0.1: ICMP echo reply, id 55286, seq 1, length 64
10:46:03.967997  In IP 192.168.1.1 > 172.16.0.1: ICMP echo reply, id 55286, seq 2, length 64
10:46:04.954291  In IP 192.168.1.1 > 172.16.0.1: ICMP echo reply, id 55286, seq 3, length 64
10:46:05.929620  In IP 192.168.1.1 > 172.16.0.1: ICMP echo reply, id 55286, seq 4, length 64
```

So, response packets are actually being sent to host-A, which is clearly an innocent victim here.

To enable uRPF strict mode in Junos, and protect host-A, just configure the `rpf-check` option under the ge-0/0/0 interface's family inet:

```
[edit]
lab@vMX-1# set interfaces ge-0/0/0 unit 0 family inet rpf-check

[edit]
lab@vMX-1# show interfaces ge-0/0/0
unit 0 {
    family inet {
        rpf-check;
        address 10.0.0.111/24;
    }
}

[edit]
lab@vMX-1# commit
commit complete
```

Now if someone starts sending spoofed packets from the direction of ge-0/0/0 interface:

```
[edit]
lab@vMX-2# run ping 192.168.1.1 routing-instance host-B source 172.16.0.1 count 5
PING 192.168.1.1 (192.168.1.1): 56 data bytes
```

```
--- 192.168.1.1 ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
```

The vMX-1 will drop these packets due not passing the uRPF check, which is reflected in the packet statistics:

```
[edit]
lab@vMX-1# run show interfaces ge-0/0/0 extensive | match RPF
     Flags: Sendbcast-pkt-to-re, uRPF
     RPF Failures: Packets: 5, Bytes: 420
```

Strict mode works by default when you enable `rpf-check`, and typically it is a way to go. But if you want to enable loose mode, just include `mode loose` option:

```
[edit]
lab@vMX-1# set interfaces ge-0/0/0 unit 0 family inet rpf-check mode loose

[edit]
lab@vMX-1# commit
commit complete
```

However, loose mode does not drop spoofed packets with the 172.16.0.1 source address in this scenario (we omitted corresponding outputs to save space). It would only drop packets with addresses that are completely missing in the routing table. For example, the 192.168.255.255 address is not in vMX-1's routing table, so the following packets:

```
lab@vMX-2# run ping 192.168.1.1 routing-instance host-B source 192.168.255.255 count 10
PING 192.168.1.1 (192.168.1.1): 56 data bytes

--- 192.168.1.1 ping statistics ---
10 packets transmitted, 0 packets received, 100% packet loss
```

…are going to be dropped by loose mode uRPF:

```
[edit]
lab@vMX-1# run show interfaces ge-0/0/0 extensive | match RPF
     Flags: Sendbcast-pkt-to-re, uRPF, uRPF-loose
     RPF Failures: Packets: 10, Bytes: 840
```

It is interesting to add a default route to vMX-1 and see if the packets still get dropped:

```
[edit]
lab@vMX-1# set routing-options static route 0/0 next-hop 10.0.0.222
[edit]
lab@vMX-1# commit
commit complete
```

Test from vMX-2's host-B instance:

```
[edit]
lab@vMX-2# run ping 192.168.1.1 routing-instance host-B source 192.168.255.255 count 10
PING 192.168.1.1 (192.168.1.1): 56 data bytes
```

```
--- 192.168.1.1 ping statistics ---
10 packets transmitted, 0 packets received, 100% packet loss
```

Statistics on vMX-1:

```
[edit]
lab@vMX-1# run show interfaces ge-0/0/0 extensive | match RPF
     Flags: Sendbcast-pkt-to-re, uRPF, uRPF-loose
     RPF Failures: Packets: 20, Bytes: 1680
```

Yes, the packets are still getting dropped because the default route for the purpose of the uRPF check is just ignored on the vMX router that uses virtual Trio-based line cards (similar to MPC cards in functionality, see a previous section for explanation of loose mode with default routes specifics).

Fail Filters

Sometimes you want to allow some traffic that would otherwise be dropped by uRPF. This may happen in an asymmetric routing scenario or with DHCP/BOOTP traffic.

In particular, if you have a DHCP or BOOTP relay configured on the router interface, these packets will always be dropped by uRPF if you enable it. This is because such packets use 0.0.0.0 source and 255.255.255.255 destination. They are just not meant to be routed in a standard way.

This is how you can avoid dropping such traffic by defining a `fail-filter`:

```
[edit]
lab@vMX-1# show firewall family inet filter rpf-dhcp
term dhcp {
   from {
      source-address {
         0.0.0.0/32;
      }
      destination-address {
         255.255.255.255/32;
      }
   }
   then accept;
}

[edit]
lab@vMX-1# show interfaces lt-0/0/10 unit 1
encapsulation ethernet;
peer-unit 0;
family inet {
   rpf-check fail-filter rpf-dhcp;
   address 172.16.0.2/24;
}
```

As you see, fail filters for uRPF are very similar to standard firewall filters.

Feasible Paths Option

One last option to consider regarding uRPF is called *feasible paths*. It is common in a routing table that there are active and non-active (backup or feasible) paths. By default, Junos only uses active paths for strict mode uRPF check. In some cases, such as in an unequal-cost asymmetric routing scenario, you may want to use non-active (feasible) paths as well.

The concept is illustrated using lab topology shown in Figure 8.8.

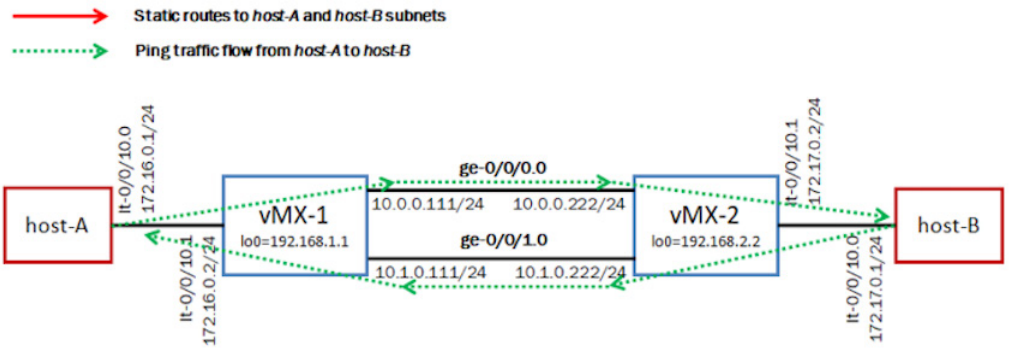


Figure 8.8 uRPF Scenario with Feasible Paths

Static routes for host-A (172.16.0.0/24) and host-B (172.17.0.0/24) subnets are configured so that routing is always asymmetric when both links (ge-0/0/0, ge-0/0/1) are up, however backup paths are also configured:

```
[edit]
lab@vMX-1# show routing-options static
route 172.17.0.0/24 {
   qualified-next-hop 10.0.0.222;
   qualified-next-hop 10.1.0.222 {
      preference 7;
   }
}

[edit]
lab@vMX-1# run show route 172.17.0.0/24

inet.0: 10 destinations, 11 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.17.0.0/24      *[Static/5] 01:29:21
                  > to 10.0.0.222 via ge-0/0/0.0
                   [Static/7] 01:29:21
                  > to 10.1.0.222 via ge-0/0/1.0
```

```
[edit]
lab@vMX-2# show routing-options static
route 172.16.0.0/24 {
   qualified-next-hop 10.0.0.111 {
      preference 7;
   }
   qualified-next-hop 10.1.0.111;
}

[edit]
lab@vMX-2# run show route 172.16.0.0/24

inet.0: 10 destinations, 11 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.0.0/24      *[Static/5] 01:28:48
                  > to 10.1.0.111 via ge-0/0/1.0
                   [Static/7] 01:28:48
                  > to 10.0.0.111 via ge-0/0/0.0
```

So far, pinging from host-A to host-B will work:

```
[edit]
lab@vMX-1# run ping 172.17.0.1 routing-instance host-A count 2
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: icmp_seq=0 ttl=62 time=1.471 ms
64 bytes from 172.17.0.1: icmp_seq=1 ttl=62 time=2.282 ms

--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.471/1.877/2.282/0.406 ms
```

Now, you enable uRPF strict mode on ge-0/0/0 and ge-0/0/1 interfaces:

```
[edit]
lab@vMX-1# set interfaces ge-0/0/0.0 family inet rpf-check

[edit]
lab@vMX-1# set interfaces ge-0/0/1.0 family inet rpf-check

[edit]
lab@vMX-1# commit
commit complete
```

After this change, the ping fails due to uRPF:

```
[edit]
lab@vMX-1# run ping 172.17.0.1 routing-instance host-A count 2
PING 172.17.0.1 (172.17.0.1): 56 data bytes

--- 172.17.0.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss

[edit]
lab@vMX-1# run show interfaces ge-0/0/1 extensive | match RPF
     Flags: Sendbcast-pkt-to-re, uRPF
     RPF Failures: Packets: 2, Bytes: 168
```

This is because you have asymmetric routing. To make uRPF not drop traffic that uses asymmetric paths, enable `unicast-reverse-path feasible-paths` option this way:

```
[edit]
lab@vMX-1# set routing-options forwarding-table unicast-reverse-path feasible-paths

[edit]
lab@vMX-1# show routing-options forwarding-table
unicast-reverse-path feasible-paths;

[edit]
lab@vMX-1# commit
commit complete
```

Now check with a ping:

```
[edit]
lab@vMX-1# run ping 172.17.0.1 routing-instance host-A count 2
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: icmp_seq=0 ttl=62 time=1.591 ms
64 bytes from 172.17.0.1: icmp_seq=1 ttl=62 time=2.320 ms

--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.591/1.955/2.320/0.365 ms
```

As you can see, uRPF now works correctly, even with asymmetric routing.

Appendix

How to Build Your Own Virtual Lab

By Chris Parker and Christian Scholz

As recently as twenty years ago, learning your craft in this industry required actual, physical, expensive hardware – and lots of it. Early devices were beasts by any metric you care to judge them by, and building your own personal lab was no trivial task. It's fair to say that engineers who managed to master the CLI against all those odds are due a lot of respect! Their electric bills alone surely made them worthy of an honorary JNCIE certification.

Engineers love having physical hardware to play with, and nowadays it's possible to buy second-hand devices via online auction sites. Some listings are legitimate, like ones from engineers who are selling their own labs to the next generation of techies. Other listings are less legitimate. If you choose to buy second-hand hardware, keep an eye out for auctions where the seller hasn't even bothered to remove the asset stickers of the company they, er, *acquired* the devices from.

But even at second-hand prices, for most of us the cost is simply too much of a barrier to entry, let alone the electric and the storage space required – and that's not to mention the noise! Oh my goodness the noise.

Luckily though, in the 2020s we have a fantastic alternative: there's a rich variety of virtualization options available to us, all completely free. You don't have to spend a single penny to master Junos; you can do it all at zero cost, from your laptop. Indeed, if you know where to look you might even be able to get some cloud storage and compute for free, at least for a short while.

In this Appendix we're going to show you three options available to those of you who are interested in setting up your own virtual lab. The first two options, GNS3 and EVE-NG, are applications that emulate actual network operating system images of almost any vendor, including Juniper. An entire book could be written on using these applications - and as you'll soon see, our dear Ambassador colleague (and co-author of this chapter) Christian Scholz actually has written a book on EVE-NG! (Chris Parker is writing this paragraph. Christian didn't just call himself a dear colleague.) (Or did he? You'll never know.)

After that, we'll introduce you to a third option: Juniper vLabs is a free web-based tool where you can play with pre-built topologies focusing on a huge variety of technologies, with the entire Junos command line at your fingertips.

GNS-3 and EVE-NG

https://www.gns3.com/
https://www.eve-ng.net/

GNS3 (Graphical Network Simulator) and EVE-NG (Emulated Virtual Network - Next Generation) are two extremely popular emulation tools. These free applications let you import actual Junos images to create your own virtual topologies. Once your device is powered on, you can connect to it via a console session, or via Telnet/SSH. Depending on the device, you may even be able to manage it via a graphical web interface.

Both GNS3 and EVE-NG are fantastic, and you can see screenshots of each below. Although each application has its own unique look and feel, they both ultimately work in similar ways on the surface: you'll notice that both applications have a menu on the screen that allows you to drop virtual machines into your topology, at which point you can drag-and-drop cables between those devices, then right-click to power them on, or capture packets on a particular interface.

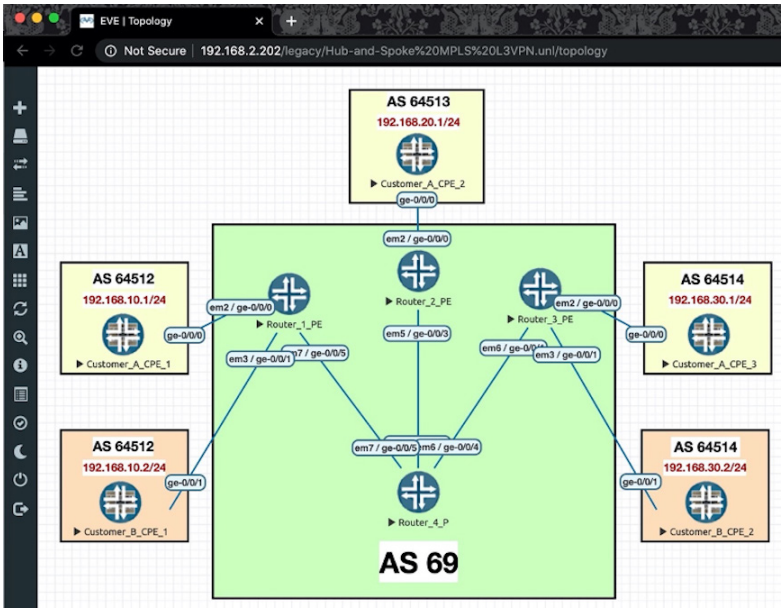


Figure A1 *EVE-NG, with a Topology Managed Through a Web Browser*

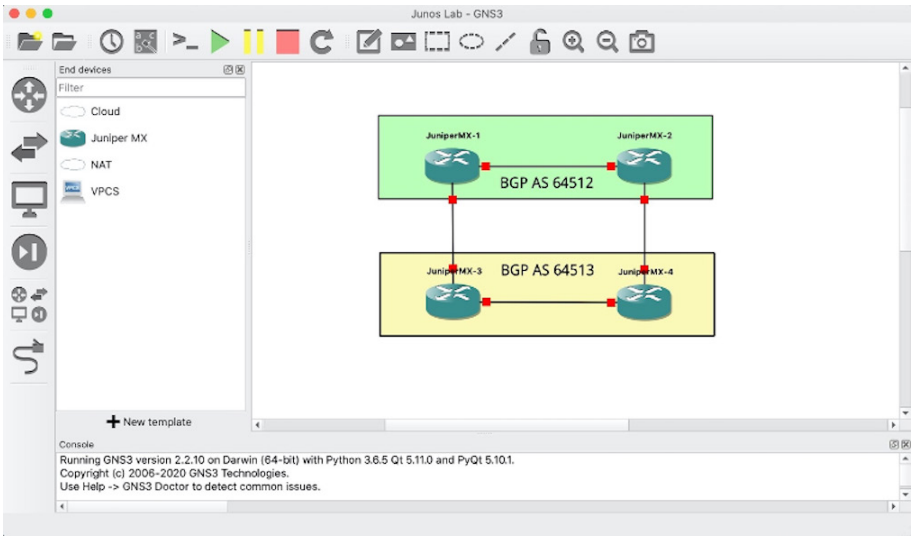


Figure A2 *A Simple Topology In the GNS3 Application*

Both GNS3 and EVE-NG also give you the ability to connect an interface on a virtual device to your actual home LAN, so you can access these devices from other machines on the network, and allow these virtual network devices to connect to the internet.

Both pieces of software have extensive documentation on their websites to talk you through the installation process for the software itself, and then guides on how to import images.

As you can tell, both applications have a lot in common! So you might be wondering what the difference is between the two.

Which is Better?

GNS3 is the oldest of the two, and as such is a very mature and robust application with a very large user base.

Both GNS3 and EVE-NG have excellent documentation to help new users, and you'll have no trouble finding step-by-step guides on installing and using them at their respective websites. However, GNS3's larger user base inevitably means that there are more blogs and YouTube videos created by the community itself.

EVE-NG was the first free emulation tool to allow you to configure your topology via a web browser from anywhere in the world. Yes – even with the poor-man's Internet from Germany. This means that you can run EVE-NG itself on a server in another room, perhaps even the other side of the world, and connect to your EVE-NG instance remotely.

This is incredibly helpful: each individual virtual appliance will take up a significant amount of RAM and CPU resource, which can put a big strain on a laptop if you want to run a topology with more than a few devices. The ability to put all the compute power onto an external server means that your own personal computer can be kept light, without forcing your laptop's fan to spin so fast that it sounds like it's about to take off into the sky.

All of these unique advantages are in flux. EVE's growth in popularity means that its community-driven documentation gets stronger every week. At the time of this writing, GNS3 has launched its own web interface which is currently in beta, as an experimental feature. The truth of it is that for the most part, the choice between the two comes down to user preference.

With that in mind, it's worth us telling you that we much prefer EVE-NG, for a few reasons.

Why Do We Personally Prefer EVE-NG?

The first, and perhaps the only reason that really matters is this: the ability to run the actual application on a separate machine has been transformative for our studies. We regularly like to lab things up while we're out and about, whether it be in a lunch break at work or at a table outside a coffee shop. The ability to just connect to our EVE server remotely with nothing more than a web browser gives the user a lot of freedom and flexibility.

In addition, we find the EVE-NG user interface more intuitive and simple, and we find the process of adding new images to be a bit more clean in EVE-NG. We've also generally found the EVE experience to be smoother, with fewer errors, and more images working first-time. The development team behind EVE is also very open for new ideas and enhancements and reacts to problems very fast. This is also true of the GNS3 team; the authors just have personal and positive experiences with the EVE developers.

We cannot stress that these are only the personal opinions and experiences of the two authors of this chapter. GNS3 is also a really excellent piece of software, and we strongly encourage you to download and try both of them.

There are actually two versions of EVE, a free Community Edition and a paid Professional Edition ($100 per year, per concurrent user, at the time of this writing) with extra features such as the ability to run multiple labs at once, and the ability to have multiple users work on the same lab at the same time. It also introduces functionality like simulating jitter and packet-loss to simulate link problems, and the ability to "hot-patch" while the devices are running. We used the free edition but do take a look at both the free and the paid versions. You may eventually choose to buy the Professional Edition just to support the community.

We mentioned earlier that Christian Scholz (co-author of this Appendix) is currently writing a *Day One* book for Juniper, focusing on setting up an EVE-NG server from scratch, and getting the most out of it once it's up and running. This mighty book is scheduled to come out by the end of the 2020 and may even have been published by the time you read these words! With that book in your hands to smoothly guide you through the entire installation process, we can confidently recommend EVE-NG as our personal favourite of the two emulation options.

Thanks to contributions from the Ambassadors, EVE constantly gets support for new Juniper images (nodes) and even supports the latest releases before GNS3. You can even run images on EVE that are not officially supported yet, even if you use a new vendor that was founded yesterday. That makes it even more amazing.

NOTE We also want to mention that you can use VMWare's ESXi to create virtual labs. A discussion of ESXi is out of scope for this book, but bear this option in mind in case you ever happen to have the chance to learn ESXi.

Where Do I Get Junos Images?

Getting Juniper images for EVE-NG and GNS is thankfully very simple, and the folks at Juniper have made it easy to access trial versions of the following products:

vSRX (Firewall): https://www.juniper.net/us/en/dm/free-vsrx-trial/

vMX (Router): https://www.juniper.net/us/en/dm/free-vmx-trial/

vQFX (Switch): https://www.juniper.net/us/en/dm/free-vqfx-trial/

To access these images you need to register a Juniper account either through your company (Partner Account) or through the registration page (for end-users) at https://userregistration.juniper.net/

After registering your account, you will be able to download the official trial images and use them with whatever emulation software you like. Each device comes with a 60-day trial license of the full-feature set.

The trial license starts when you power-up the device for the first time. A smart user could save their config after 59 days, "delete" the image (i.e. wipe it in EVE-NG), then start a new trial and insert the license again. While this may seem perfectly fine for a lab, we would never encourage you to do this, and we would, of course, never tell you that you have unlimited time throughout this process. Again: we would never suggest that you do that.

With these three different device types available to you, you will be able to create amazing labs made up of routers, switches, and firewalls.

After the device is powered up in EVE-NG or GNS3, you can download the actual license by going to Step 2 of the download process on the pages we linked to above. With the `request system license add terminal` command you can install your license on your lab device, and later check with `show system license` if it has been installed correctly.

Juniper vLabs

https://vlabs.juniper.net/

Some folks don't have a computer powerful enough to run so many virtual devices. Others may be very new to networking and would like pre-configured topologies so they can play with a working base configuration, observe what things look like when everything's correct, and then break it or build on it to see what happens.

For those engineers, Juniper created vLabs: an ever-increasing series of topologies that have been architected for you in advance, featuring products as diverse as the vMX, vQFX, vSRX, Healthbot, Northstar, cSRX, Sky Enterprise, and more. There are labs with multi-area OSPF, multi-AS BGP, IPsec VPNs, and even labs for you to play with Python and PyEZ for network automation.

Every month the number of pre-made labs grows. For example, at the time of this writing Juniper added a VLAN lab a month ago, a Northstar lab a fortnight ago, and an EVPN-VXLAN lab this week. By the time you read this, there will no doubt be even more.

Once you sign in with your Juniper Account you'll be shown the list of labs you can use. At the top of the vLabs page you'll see standalone boxes, which are handy for quickly trying out some commands. Scroll down further and you'll see the good stuff: full-scale labs, as shown in Figure A3.

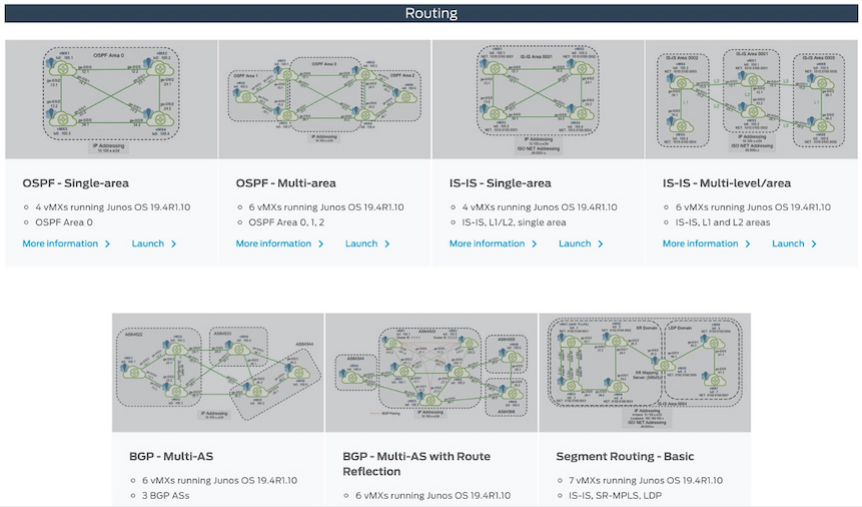


Figure A3 *A Small Selection of the Many Topologies on Offer at the vLabs Website*

Choose your lab to see the details, as in Figure A4. You'll see exactly what you're getting, along with a few things you might like to try once you're in.

Note that at the bottom of Figure A4 it says you can also do anything you like. Remember, these VMs feature the full Junos command set, which means that you could theoretically spin up a BGP topology, delete all the BGP config, and put something like multicast on there instead! You can use these "physical" topologies in any way you like, to learn any protocol you like. Don't feel limited to what comes pre-configured. If the VM can support it, you can do it.

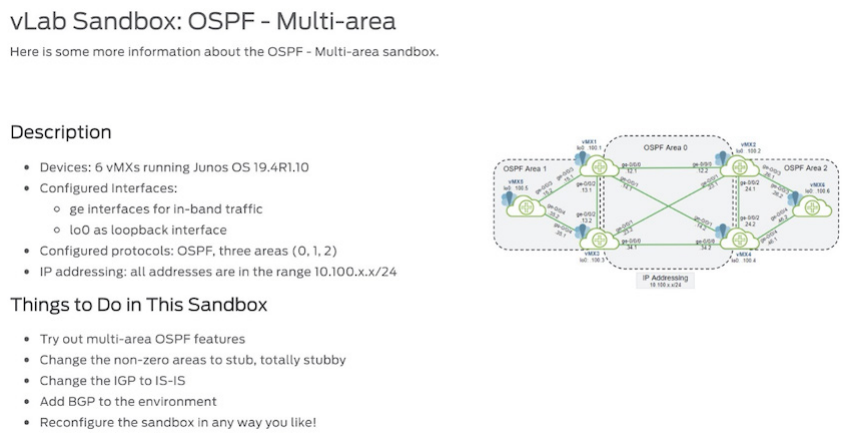


Figure A4 *An Example of a Preview You'll See Before Loading a Lab*

Once you're in the lab itself you'll see a Reserve button in the top-right corner. Click it, and you'll see it change to "Setup," like in Figure A5. At this stage, go make a cup of coffee. It will take about 20 minutes for the lab to come live. Remember, these are all actual VMs that are being created in the cloud at your request.

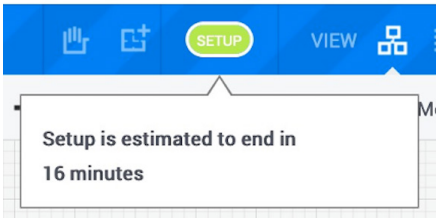


Figure A5 *The "Setup" Button Within the Lab*

Once the lab has been created you'll see a screen like the one in Figure A6. Notice the button in the top-right now says "Active." Each of the routers are displayed as a white rectangle, and you'll notice the arrows between them, which are annotated with the name of the interface linking them together. On the left you'll see that we hovered over a vMX to bring up a list of options on how to manage it.

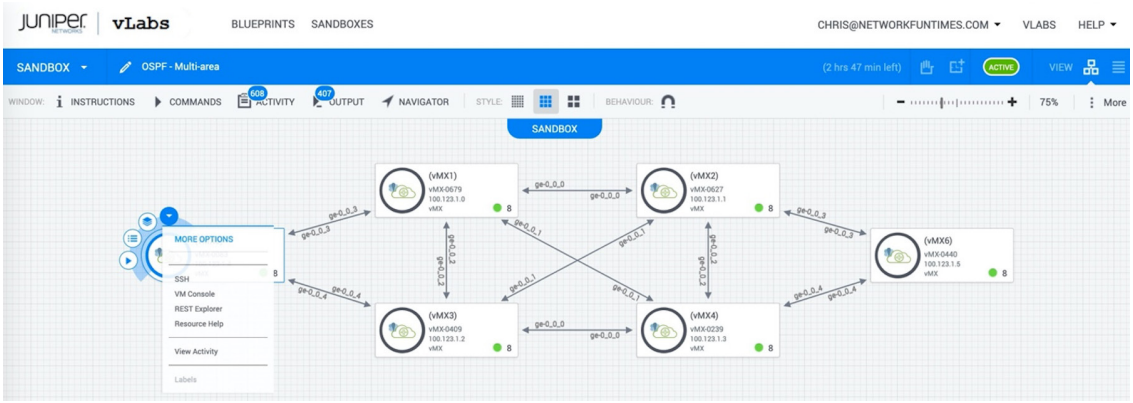


Figure A6 *A View of the Topology Within the Lab*

If you choose SSH then you'll see a new tab pop open with a full Junos CLI, like in Figure A7. You can check the config, check the neighborships/adjacencies, add to the config, remove the config, and do absolutely anything you like.

```
Last login: Fri Jan 24 22:51:37 2020 from 66.129.235.126
--- JUNOS 19.4R1.10 Kernel 64-bit  JNPR-11.0-20191115.14c2ad5_buil
jcluser@vMX5> show configuration protocols ospf
area 0.0.0.1 {
    interface ge-0/0/3.0;
    interface ge-0/0/4.0;
    interface lo0.0;
}

jcluser@vMX5> show ospf neighbor
Address          Interface              State           ID               Pri  Dead
10.100.15.1      ge-0/0/3.0             Full            10.100.100.1     128    36
10.100.35.1      ge-0/0/4.0             Full            10.100.100.3     128    35

jcluser@vMX5>
```

Figure A7 *A vMX CLI from Juniper vLabs. Notice That Both OSPF Neighbors are Up.*

Something that isn't immediately made clear is that this CLI window actually runs via Guacamole, which is a remote desktop client that runs within HTML5. This is handy to know, because you might initially think that things like copy-paste don't quite work. But knowing that the Junos CLI is displaying within Guacamole means you can hit Ctrl + Alt + Shift to bring up the Guacamole Menu for all your copying needs. If you're not familiar with Guacamole as a service, spend a few minutes reading around it on the internet, so you can get the most out of your vLabs session. And be sure to read the vLabs User Guide and FAQ too. There are some extra features that will make your life a lot easier. Don't do the usual engineer thing of jumping in without reading the instructions!

The labs run for three hours by default, though you can extend up to six hours if you're deep into studying. And of course, just before your time runs out you can save your configs to a text editor, then spin up another lab and just put the configs back on again.

Summary

In some ways, it has never been harder to learn networking. There's always new protocols to learn, and while it's true that old protocols fade away, they certainly don't fade away at the same pace that new protocols come along. Add to that the fact that we also need to know scripting, automation tools, Linux, multiple vendors, and plenty more besides. It can be overwhelming!

On the other hand, it has never been easier to learn networking. Unlike the early pioneers of our trade, we have access to a wealth of books, blog posts, YouTube tutorials, vendor courses, communities on Twitter and Slack and Discord – and as you've now seen, you can choose from a wide variety of virtualisation options which allow you to learn it all for free, in your spare time, at home. Stop to acknowledge the engineers who came before us, who had to stay behind after work just to get access to the ten-quadrillion-dollar lab in their employer's poorly ventilated basement.

Take some time to read up on GNS3, EVE-NG, and vLabs. All three options offer their own unique advantages and use cases, and you should use the one you find the most comfortable and enjoyable. Good luck with your labs, and remember to look out an up-coming Day One book on EVE-NG!

Other Day One books by Juniper Ambassadors

Day One: Ambassadors' Cookbook for 2019

Day One: Inside the MX 5G

Day One: Automating Junos with Salt

Day One: Junos for IOS Engineers

Day One: Juniper Ambassadors' Cookbook 2018

Day One: Juniper Ambassadors' Cookbook 2017

Day One: Juniper Ambassadors' Cookbook 2014

Day One: Juniper Ambassadors' Cookbook for Enterprise

Day One: MPLS Up and Running On Junos

Day One: Finishing Junos Deployments

Day One: Routing the Internet Protocol

Day One: vMX Up and Running

This Week: An Expert Packet Walkthrough on the MX Series 3D

Day One: MPLS for Enterprise Engineers

Available at https://www.juniper.net/dayone.