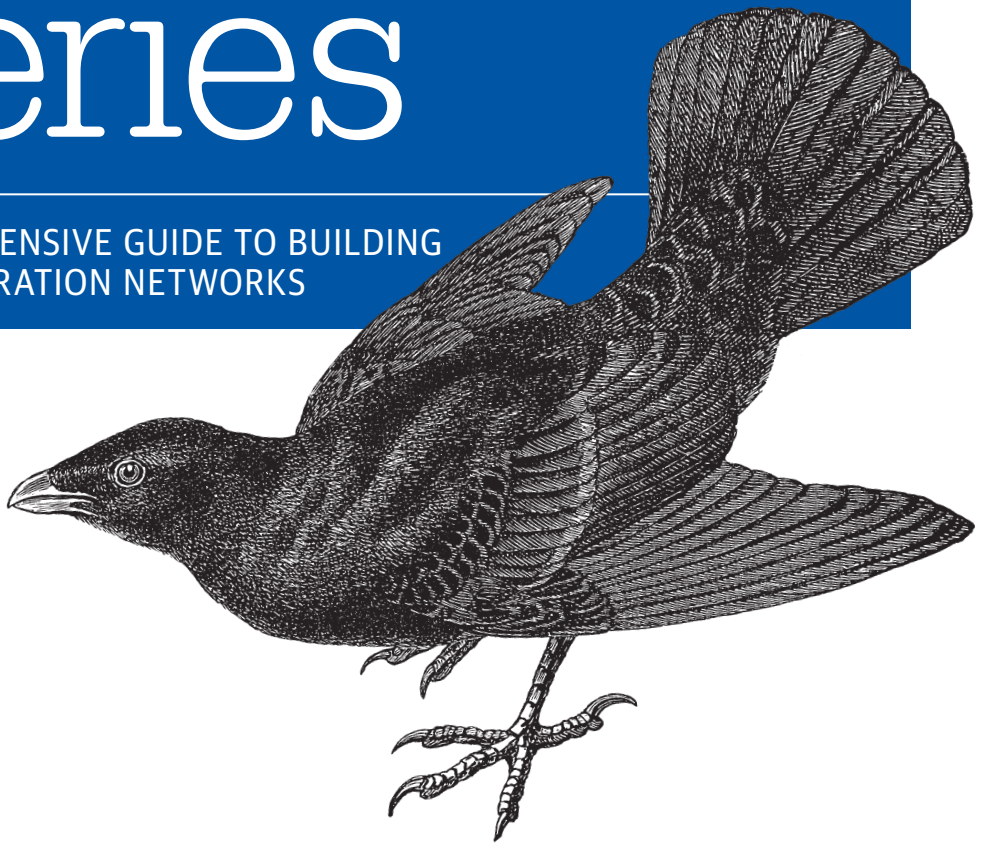
O'REILLY®

Juniper QFX5100 Series

A COMPREHENSIVE GUIDE TO BUILDING NEXT-GENERATION NETWORKS



JUNIPER NETWORKS®

Douglas Richard Hanks, Jr.

O'REILLY®

Juniper QFX5100 Series

Ideal for network engineers involved in building a data center, this practical guide provides a comprehensive and technical examination of the new Juniper QFX5100 switching family. You'll learn how the Juniper QFX5100 enables you to create simple-to-use data centers or build some of the largest IP Fabrics in the world.

This book is chock-full of helpful technical illustrations and code examples to help you get started on all of the major architectures and features of Juniper QFX5100 switches, whether you're an enterprise or service provider. With this book, you'll be well on your way to becoming a Juniper QFX5100 expert.

All of the examples and features are based on Junos releases 13.2X51-D20.2 and 14.1X53-D10.

- Fully understand the hardware and software architecture of the Juniper QFX5100
- Design your own IP Fabric architecture
- Perform in-service software upgrades
- Be familiar with the performance and scaling maximums
- Create a data center switching fabric with Virtual Chassis Fabric
- Automate networking devices with Python, Ruby, Perl, and Go
- Build an overlay architecture with VMware NSX and Juniper Contrail
- Export real-time analytics information to graph latency, jitter, bandwidth, and other features

"If you plan to deploy QFX5100 in your data center (and you should if you're a Juniper shop), you should read this book before starting your design process, and it's a definite must-read before the implementation."

—Ivan Pepelnjak
www.ipSpace.net and *blog.ipSpace.net*

"Required reading for anyone about thinking about deploying the Juniper QFX5100 series. A deep dive into both the hardware and software aspects and real world scenarios makes this an essential read."

—Darren O'Connor
Dual CCIE #38070 and JNCIE-SP #2227

Douglas Richard Hanks, Jr., Chief Architect at Juniper Networks, works on next-generation switching products and solutions. Aside from being the founder of the Bay Area Juniper Users Group (BAJUG), Doug is the author of O'Reilly's bestselling *Juniper MX Series*, as well as several books published by Juniper Networks.

NETWORKING

US $59.99 **CAN $62.99**

ISBN: 978-1-491-94957-3

Twitter: @oreillymedia
facebook.com/oreilly

Juniper QFX5100 Series

Douglas Richard Hanks, Jr.

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Juniper QFX5100 Series
by Douglas Richard Hanks, Jr.

Printed in the United States of America.

Published by O'Reilly Media, Inc. , 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Mike Loukides and Allyson MacDonald
Production Editor: Nicole Shelby
Copyeditor: Octal Publishing Services
Proofreader: Charles Roumeliotis
Indexer: Lucie Haskins
Interior Designer: David Futato
Cover Designer: Ellie Volckhausen
Illustrator: Douglas Richard Hanks, Jr.

December 2014: First Edition

Revision History for the First Edition
2014-11-17: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491949573* for release details.

978-1-491-94957-3

[LSI]

Dedicated to my wife and my parents. You guys are the best.

Love,
Douglas

Table of Contents

Preface

The Juniper QFX5100 series of switches is quickly becoming the go-to platform for a wide variety of customers who love and cherish virtualization, whether it's compute virtualization with VMware vSphere, Microsoft Hyper-V, or Linux KVM; cloud virtualization with OpenStack, CloudStack, or IBM SmartCloud; or network virtualization with Contrail or NSX. Virtualization is driving the need for high-density 10GbE access and overlay networking so that compute, storage, and network can be decoupled from the physical hardware and fully orchestrated from a single pane of glass.

The Juniper QFX5100 family was designed from the ground up to solve the challenges of high-density 10GbE and overlay networking. These switches support high-density 10GbE and 40GbE interfaces, 550 nanoseconds of latency, and have hardware support for overlay networks such as Virtual Extensible LAN (VXLAN) and Network Virtualization using Generic Routing Encapsulation (NVGRE). Each customer designs and operates their network differently; Juniper QFX5100 switches embrace this concept and don't force you to use a specific proprietary protocol. The QFX5100 series supports the following six modes to suit your specific needs:

- Standalone
- QFabric node
- Virtual Chassis Fabric (VCF)
- Multi-Chassis Link Aggregation (MC-LAG)
- IP Fabrics (Clos)
- Virtual Chassis

The Juniper QFX5100 family also takes virtualization to heart: under the hood it uses Linux and KVM to virtualize the network operating system—Junos—to reap all the benefits of virtualization, such as snapshots and In-Service Software Upgrades (ISSU).

This book shows you, step by step, how to build a better network using the Juniper Juniper QFX5100 Series—it's such a versatile platform that it can be placed in the core, aggregation, or access of any type of network and provide instant value. Juniper QFX5100 switches were designed to be network virtualization beasts. You can choose between six different networking technologies and support overlay networking directly in hardware with no performance loss.

No Apologies

I'm an avid reader of technology books, and I always get a bit giddy when a new book is released because I can't wait to read it and learn more about a specific technology. However, one trend I have noticed is that every networking book tends to regurgitate the basics over and over. There are only so many times you can force yourself to read about spanning tree, the split-horizon rule, or OSPF LSA types. One of the goals of this book is to introduce new and fresh content that hasn't been published before.

I made a conscious decision to keep the technical quality of this book very high; this created a constant debate whether to include primer or introductory material in the book to help refresh a reader's memory with certain technologies and networking features. In short, here's what I decided:

Spanning Tree and Switching
: Spanning tree and switching is covered in great detail in every JNCIA and CCNA book on the market. If you want to learn more about spanning tree or switching, check out *Junos Enterprise Switching* (O'Reilly), or *CCNA ICND2 Official Exam and Certification Guide*, Second Edition (Cisco Press).

Routing Protocols
: There are various routing protocols such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) used throughout this book in case studies. No introductory chapters are included for IS-IS or OSPF; I'm making the assumption that you have read *Junos Enterprise Routing*, Second Edition (O'Reilly) or *Juniper Networks Certified Internet Expert Study Guide* (Juniper Networks), which cover these topics thoroughly. If you haven't read them, I highly recommend that you do so.

Virtual Chassis
: I was really torn on this subject. Yes, the Juniper QFX5100 series supports Virtual Chassis, and it's great, but Virtual Chassis has been covered in depth in other books such as *Juniper MX Series* and *Junos Enterprise Switching*. Do we really need another chapter on Virtual Chassis? My response is no. I don't want to devalue the benefits of Virtual Chassis, but there are already other sources of information out there that cover it in enough detail that I don't need to write another chapter about it. Does the QFX5100 do Virtual Chassis a bit differently?

Are there any caveats? Yes and yes. I will specifically address these questions in Chapter 3. Don't worry, I make up for this by adding a chapter specifically on Virtual Chassis Fabric (Chapter 5).

Multi-Chassis Link Aggregation
: Ah, MC-LAG we meet again. I spent a great deal of time on MC-LAG in *Juniper MX Series*, and I'm faced with the same question here: do we need yet another chapter on MC-LAG? My answer is no. If you want to learn more about MC-LAG, please read *Juniper MX Series*. It explains thoroughly MC-LAG and is complete with a case study. Does the QFX5100 series have any caveats? Yes, a few, which I discuss specifically in Chapter 3.

Quality of Service
: Classifiers, schedulers, and drop profiles. Oh My! If you want to learn more about Quality of Service (QoS), check out *Juniper MX Series*; it covers QoS in depth. The QFX5100 series has a different buffer management system than other platforms, which is covered in Chapter 1.

After many hours of debate, I decided that I should defer to other books when it comes to introductory material and keep the content of this book at an expert level. I expect that most readers already have their JNCIE or CCIE (or are well on their way) and will enjoy the technical quality of this book. For novices, I want to share an existing list of books that are widely respected within the networking community:

- *Juniper MX Series* (O'Reilly) (the best book out of the bunch)
- *Junos Enterprise Routing*, Second Edition (O'Reilly)
- *Junos Enterprise Switching* (O'Reilly)
- *QoS-Enabled Networks* (Wiley & Sons)
- *MPLS-Enabled Applications*, Third Edition (Wiley & Sons)
- *Network Mergers and Migrations* (Wiley)
- *Juniper Networks Certified Internet Expert* (Juniper Networks)
- *Juniper Networks Certified Internet Professional* (Juniper Networks)
- *Juniper Networks Certified Internet Specialist* (Juniper Networks)
- *Juniper Networks Certified Internet Associate* (Juniper Networks)
- *CCIE Routing and Switching*, Fourth Edition (Cisco Press)
- *Routing TCP/IP*, Volume I and II (Cisco Press)
- *OSPF and IS-IS* (Addison Wesley)
- *OSPF: Anatomy of an Internet Routing Protocol* (Addison Wesley)
- *The Art of Computer Programming* (Addison Wesley)

- *TCP/IP Illustrated*, Volume 1, 2, and 3 (Addison Wesley)
- *UNIX Network Programming*, Volume 1 and 2 (Prentice Hall PTR)
- N*etwork Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices* (Morgan Kaufmann)

What's in This Book?

This book was written for network engineers, by a network engineer. The ultimate goal of this book is to share with you the logical underpinnings of the Juniper QFX5100 family of switches. Each chapter represents a specific vertical within the Juniper QFX5100 series and will provide enough depth and knowledge to give you the confidence to implement and design new architectures for your network by using the Juniper QFX5100 series.

Here's a short summary of the chapters and what you'll find inside:

Chapter 1, Juniper QFX5100 Architecture
: In this chapter, you learn a little bit about the history and what challenges prompted the creation of the Juniper QFX5100 series. Junos is the "secret sauce" that's common throughout all of the hardware; this chapter takes a deep dive into the control plane and explains some recent important changes to the release cycle and support structure of Junos. The stars of the chapter are, of course, the Juniper QFX5100 switches; the chapter thoroughly explains all of the components such as the different platforms, modules, and hardware architecture.

Chapter 2, Control Plane Virtualization
: If you build something from the ground up to solve the challenges of virtualization, you had better take virtualization seriously. Learn how the Juniper QFX5100 series uses Linux, QEMU, and KVM to virtualize the networking operating system (Junos) and enjoy all of the benefits of virtualization such as snapshots and In-Service Software Upgrades (ISSU). But ISSU requires two routing engines! Well then, it's a good thing that we can just spin up another instance of Junos in a VM.

Chapter 3, Performance and Scaling
: All of these features are great, but I need to know the performance and scaling attributes. No problem. Chapter 4 takes a deep-dive into the control plane and data plane and explores both the physical and logical performance and scaling abilities of the Juniper QFX5100 family. You're going to like what you see.

Chapter 4, One Box, Many Options
: The Juniper QFX5100 family acknowledges that each customer designs and operates the network differently to suit their business and operational needs. One of

the crowning achievements of the Juniper QFX5100 series is that it gives you the freedom to choose between six different networking technologies and doesn't force you to use a proprietary protocol to operate your network. Read this chapter to learn more about how you can use Juniper QFX5100 switches in (1) standalone, (2) Virtual Chassis, (3) QFabric Node, (4) Virtual Chassis Fabric, (5) MC-LAG, and (6) CLOS. As the chapter's title states: one box, many options.

Chapter 5, Virtual Chassis Fabric
: Do you ever wish Ethernet fabrics were as simple as plug-n-play? Virtual Chassis Fabric (VCF) brings all of the benefits of an Ethernet fabric, but simplifies the installation and operation so that the fabric acts as a single switch. Do you need to expand the fabric? Just plug in another switch. It just works.

Chapter 6, Network Automation
: The ratio of network engineers to devices is going up. The only way to manage a large set of networking devices is through automation. This chapter shows what types of network automation the Juniper QFX5100 series supports. I'll give you a hint: it supports every major programming language, including Go.

Chapter 7, IP Fabrics (Clos)
: Charles Clos created multistage circuit switching in 1953. Let's learn how to apply his principles in networking and build large Ethernet fabrics. When it comes to a Clos fabric, scale is king.

Chapter 8, Overlay networking
: Here we explain how overlay networking decouples the network from the physical hardware. It's the driving force behind Contrail and NSX. Learn how Juniper QFX5100 switches support overlay networking in both the control plane and data plane to bridge the gap between virtualized and nonvirtualized hosts in an overlay architecture.

Chapter 9, Network Analytics
: Why is my application slow? Am I meeting my service level requirements? What's a microburst? Chapter 9 reveals that the Juniper QFX5100 has both software and hardware support for gathering detailed data on both queue and traffic statistics.

Appendix A, Under the Hood
: What's the first thing you do when you examine a car that you're interested in buying? You pop the hood and see what's powering it! Well... at least I do. <Insert Tim Allen grunt here> In this chapter, we pop the hood of the Juniper QFX5100 software stack and see what makes it tick.

Appendix B, Optical Guide
: One of the frustrations experienced by every network engineer is trying to determine what optics to use. Do you want copper, fiber, or DAC? Single-mode or

multimode fiber? Appendix B contains a great summary in table format that clearly shows you the characteristics of each optic. Finally, we summarize a product compatibility table that lists which optics are supported on each of the switching platforms.

Appendix C, BGP and VTEP Configurations
: In Chapter 7 and Chapter 8, we had great laboratories going through the deep-dive of IP fabrics and overlay networking. Appendix C presents the full configurations of the switches to show you all of the BGP and VTEP details.

Each chapter includes a set of review questions on the topics that have been covered, all designed to get you to think about and digest what you've just read. If you're not in the certification mode, the questions will provide a mechanism that facilitates critical thinking, potentially prompting you to locate other resources to further your knowledge.

This book also includes a lot of configurations and data. You may download them from GitHub at *https://github.com/Juniper/qfx5100-book*.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, and the output from commands

`Constant width bold`
: Shows commands and other text that should be typed literally by the user, as well as important lines of code

`Constant width italic`
: Shows text that should be replaced with user-supplied values

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your own configuration and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the material. For example, deploying a network based on actual configurations from this book does not require permission. Selling or distributing a CD-ROM of examples from this book does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of sample configurations or operational output from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN, for example: "*Juniper QFX5100 Series*, by Douglas Richard Hanks, Jr. Copyright 2015, Douglas Richard Hanks, Jr., 978-1-491-94957-3."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

> As with most deep-dive books, you will be exposed to a variety of hidden, Junos shell, and even VTY commands performed after forming an internal connection to a PFE component. And as always, the standard disclaimers apply.
>
> In general, a command being hidden indicates that the feature is not officially supported in that release. You should only use such commands in production networks after consultation with Juniper Networks' Technical Assistance Center (JTAC). Likewise, the shell is not officially supported or documented. The commands available can change, and you can render a router unbootable with careless use of shell commands. The same holds true for PFE component-level shell commands, often called VTY commands. Again, these are undocumented, and capable of causing network disruption, or worse, damage to the routing platform that can render it inoperable.

The hidden and shell commands that are used in this book were selected because they were the only way to illustrate certain operational characteristics or the results of complex configuration parameters.

Again, you should use hidden and shell commands only under JTAC guidance; this is especially true when dealing with a router that is part of a production network.

You have been duly warned.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

(800) 998-9938 (in the United States or Canada)

(707) 829-0515 (international or local)

(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at: *http://bit.ly/juniper_qfx5100*.

To comment or ask technical questions about this book, send email to: *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*.

Follow us on Twitter: *http://twitter.com/oreillymedia*.

Watch us on YouTube: *http://www.youtube.com/oreillymedia*.

Safari® Books Online

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

CHAPTER 1

Juniper QFX5100 Architecture

Let's start with a little bit of history to explain the problem and demonstrate the need of the Juniper QFX5100. It all starts in 2008, when Juniper Networks decided to officially enter the data center and campus switching market. Juniper released its first switch, the EX4200, a top-of-rack (ToR) switch that supports 48 1-Gigabit Ethernet (GbE) ports and 2 10GbE interfaces. The solution differentiation is that multiple switches can be connected together to create a virtual chassis: single point of management, dual routing engines, multiple line cards, but distributed across a set of switches. Juniper released its first 10GbE ToR switch running Junos in 2011, the Juniper EX4500, which supports 48 10GbE ports. The Juniper EX4200 and EX4500 can be combined to create a single virtual chassis that can accommodate a mixed 1GBE and 10GBE access tier.

More than four years in the making, Juniper QFabric was released in 2011. QFabric is a distributed Ethernet fabric that employs a spine-and-leaf physical topology, but is managed as a single, logical switch. The solution differentiation is that the core, aggregation, and access data center architecture roles can now be collapsed into a single Ethernet fabric that supports full Layer 2 and Layer 3 services. The QFabric solution comes in two sizes: the Juniper QFX3000-M scales up to 768 10GbE ports and is often referred to as the "micro fabric"; and the much larger Juniper QFX3000-G scales up to 128 ToR switches and 6,144 10GbE ports.

The data center is continuing to go through a fundamental shift to support higher speed interfaces at the access layer. This shift is being driven largely by *compute virtualization*. The shift is seen across multiple target markets. One of the biggest factors is the adoption of the cloud services offered by service providers; however, enterprise, government agencies, financial, and research institutions are adopting compute virtualization and seeing the same need for high-speed interfaces.

Specifically, the shift is happening from 1GbE to 10GbE interfaces in the access layer. To support high-density 10GbE interfaces, the core and aggregation layers need to support even higher-speed interfaces such as 40GbE to maintain a standard oversubscription of 3:1. Another trend is that storage and data are becoming collapsed onto the same network infrastructure. Whether it's via Fibre Channel over Ethernet (FCoE), Internet Small Computer System Interface (iSCSI), or Network File System (NFS), converging storage on the data network further increases the port density, speed, and latency requirements of the network.

Software-Defined Networking

Over the past couple of years, an architecture called Software-Defined Networking (SDN) was created, refined, and is taking shape, as shown in Figure 1-1. One aspect of SDN is that it makes the network programmable. OpenFlow provides an API to networking elements so that a centralized controller can precalculate and program paths into the network. One early challenge with SDN was how to approach compute and storage virtualization and provide full integration and orchestration with the network. VXLAN introduced a concept that decouples the physical network from the logical network. Being able to dynamically program and provision logical networks, regardless of the underlying hardware, quickly enabled integration and orchestration with compute and storage virtualization. A hypervisor's main goal is to separate compute and storage resources from the physical hardware and allow dynamic and elastic provisioning of the resources. With the network having been decoupled from the hardware, it was possible for the hypervisor to orchestrate the compute, storage, and network.

What's particularly interesting about the data in Figure 1-1 is that between 2007 and 2011, the progression of SDN was largely experimental and a topic of research; the milestones are evenly spaced out roughly every 10 months. Starting in 2011 the timeline becomes more compressed and we start to see more and more milestones in shorter periods of time. In a span of three years between 2011 and 2013, there are 13 milestones, which is 320 percent more activity than the first four years between 2007 and 2010. In summary, the past four years of SDN progression has been extremely accelerated.

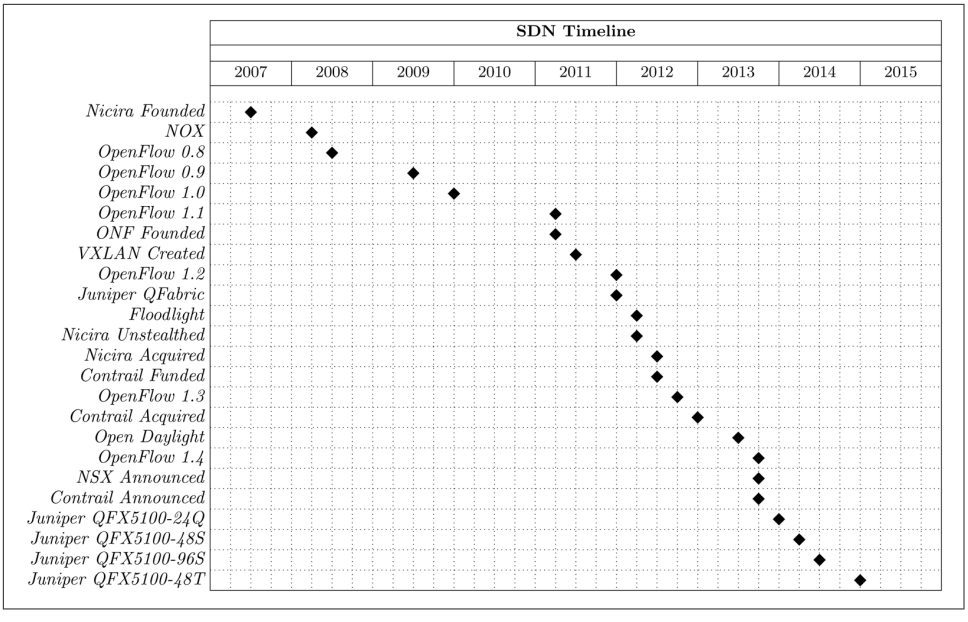


Figure 1-1. SDN milestones and introduction of the Juniper QFX5100 family

Although there was a lot of hype surrounding SDN as it was evolving, as of this writing one of the ultimate results is that there are two tangible products that bring the tenets of SDN to life: Juniper Contrail and VMware NSX. These two products take advantage of other protocols, technologies, and hardware to bring together the complete virtualization and orchestration of compute, storage, and network in a turnkey package that an engineer can use to easily operate a production network and reap the benefits of SDN.

Do you want to learn more about the architecture of SDN? For more in-depth information, check out the book *SDN: Software Defined Networks* by Thomas Nadeau and Ken Gray (O'Reilly). It contains detailed information on all of the protocols, technologies, and products that are used to enable SDN.

Juniper Contrail and VMware NSX rely on an underlying technology called *overlay networking*; this is the concept of decoupling the network from the physical hardware. One of the key technologies that enable overlay networking is VXLAN, which is a simple UDP encapsulation that makes it possible for Layer 2 traffic to traverse a Layer 3 network between a set of end points (see Figure 1-2). The overlay networks are terminated on each hypervisor, which means that the hypervisor is responsible for the encapsulation and decapsulation of the traffic coming to and from the virtual machines. The hypervisor must handle MAC address learning so that it knows which

remote host to send the encapsulated traffic to, which is based on the destination MAC address of the virtual machine.

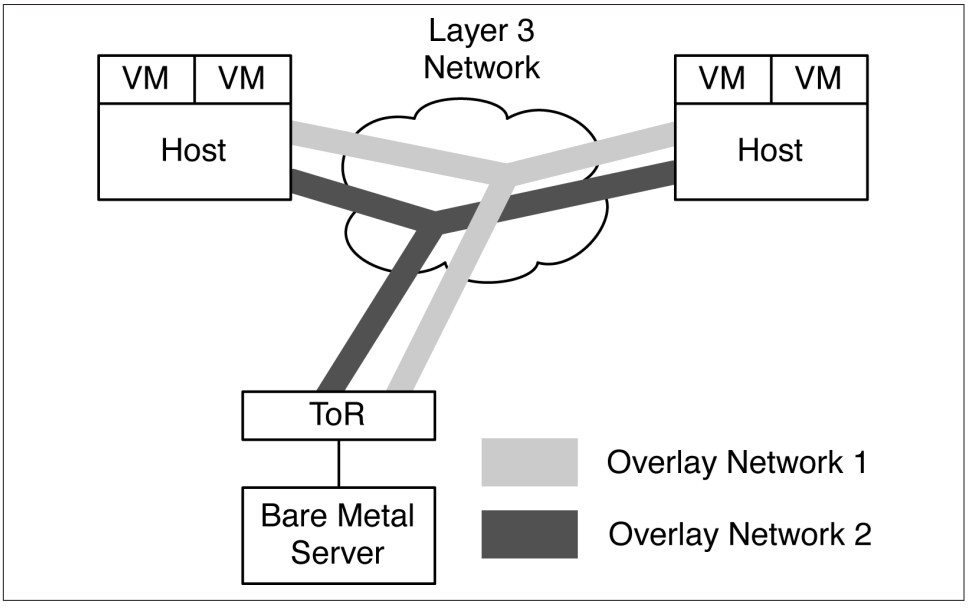


Figure 1-2. Overlay networking

The astute reader might have noticed in Figure 1-2 that in addition to supporting more than one overlay network, there are also virtualized and nonvirtualized workloads being connected by the overlay networks. In a production environment, there will always be a use case in which not all of the servers are virtualized but still need to communicate with virtual machines that are taking advantage of overlay networking. If the hypervisor is responsible for MAC address learning and termination of the overlay networks, this creates a challenge when a virtual machine needs to communicate with a bare-metal server. The solution is that the ToR switch can participate in the overlay network on behalf of nonvirtualized workloads, as is demonstrated in Figure 1-2. From the perspective of the bare-metal server, nothing changes; it sends to and receives traffic from the ToR, but the difference is that the ToR is configured to include the bare-metal server in the overlay network. With both the hypervisor and ToR handling all of the MAC address learning and overlay network termination, virtual machines and bare-metal servers are able to communicate and take full advantage of the overlay networks, such as those presented in Chapter 8.

The continued drive for high-density 10GbE access together with the evolutions of SDN and overlay networking are the key driving factors behind the introduction of the Juniper QFX5100 family in November 2013. Figure 1-3 illustrates that it is a set of data center Ethernet switches that can be used in the core, aggregation, and access

tiers of the network. The Juniper QFX5100 was specifically designed to solve the high port density requirements of cloud computing and enable nonvirtualized workloads to participate in an overlay network architecture.

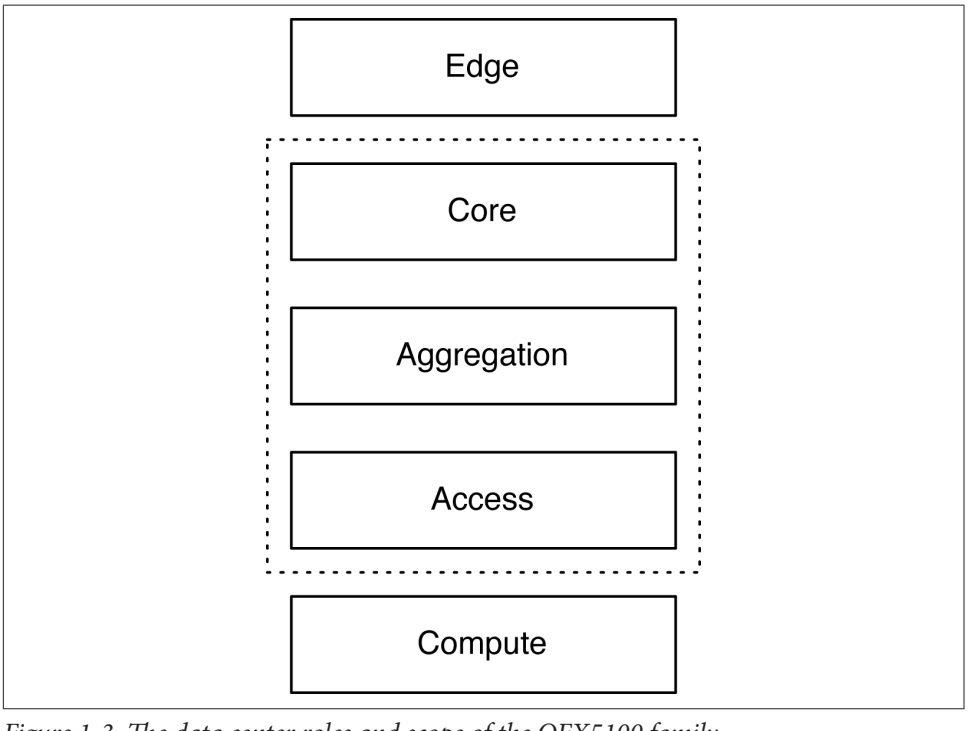


Figure 1-3. The data center roles and scope of the QFX5100 family

Having been specifically designed to solve cloud computing and SDN requirements, the Juniper QFX5100 family solves a wide variety of challenges and offers many unique benefits.

Transport
: Dense 10GbE and 40GbE interfaces to build a deterministic spine-and-leaf topology with an option of 1:1, 3:1, or 6:1 over-subscription.

Interfaces
: Each 10GbE interface is tri-speed and supports 100Mbps, 1GbE, or 10GbE. In addition, each interface can support either copper or fiber connectivity. Higher interface speeds such as 40GbE can be broken out into four 10bGE interfaces by using a breakout cable.

Overlay Networking
: Each switch offers complete integration with Juniper Contrail and VMware NSX to support overlay networking. The Juniper QFX5100 family can be configured as an end point in an overlay network architecture to support bare-metal servers.

Latency
: An intelligent algorithm is used for each ingress packet to determine which forwarding architecture—store-and-forward or cut-through—should be used to guarantee the least latency. On average the port-to-port latency is only 600 to 800 nanoseconds.

Flexible Deployment Options
: The Juniper QFX5100 doesn't force you to deploy a particular technology or proprietary protocol. It supports standalone, Virtual Chassis, QFabric node, Virtual Chassis Fabric, Multi-Chassis Link Aggregation (MC-LAG), or an IP Fabric architecture.

QFabric Node
: The Juniper QFX5100 can be used as a node in the QFabric architecture. All of the benefits of the Juniper QFX5100 are available when used as a QFabric node: higher port density, overlay networking, and lower latency.

Virtualized Control Plane
: The Juniper QFX5100 takes virtualization to heart. The control plane uses an Intel Sandy Bridge CPU. The host operating system is Linux running KVM and QEMU for virtualization. The network operating system (Junos) runs as a virtual machine and is able to take advantage of all of the benefits of virtualization such as In-Service Software Upgrade (ISSU).

Unified Forwarding Table
: Whether you need to support more MAC addresses or IPv4 prefixes in an IP Fabric architecture, with the Juniper QFX5100, you can adjust the profile of the forwarding table. There are five preconfigured profiles that range between L2 heavy to L3 heavy.

Network Analytics
: Some applications are sensitive to microbursts and latency. The Juniper QFX5100 allows you to get on-box reporting of queue depth, queue latency, and microburst detection to facilitate and speed up the troubleshooting process.

Lossless Ethernet
: When converging storage and data, it's critical that storage be handled in such a way that no traffic is dropped. The Juniper QFX5100 supports DCBX, ETS, and PFC to enable transit FCoE or lossless Ethernet for IP storage.

Virtual Chassis Fabric
: Ethernet fabrics provide the benefit of a single point of management, lossless storage convergence, and full Layer 2 and Layer 3 services. The Juniper QFX5100 can form an Ethernet fabric called a Virtual Chassis Fabric (VCF). This is a spine-and-leaf topology that supports full Equal-Cost Multipath (ECMP) routing but with all of the benefits of an Ethernet fabric.

Inline Network Services
: Traditionally, network services such as Generic Routing Encapsulation (GRE) and Network Address Translation (NAT) are handled by another device such as a router or firewall. The Juniper QFX5100 can perform both GRE and NAT in hardware without a performance loss. The Juniper QFX5100 also offers inline VXLAN termination for SDN and supports real-time networking analytics.

The Juniper QFX5100 family brings a lot of new features and differentiation to the table when it comes to solving data center challenges. Because of the wide variety of features and differentiation, you can integrate the Juniper QFX5100 into many different types of architectures.

High-Frequency Trading
: Speed is king when it comes to trading stocks. With an average port-to-port latency of 550 nanoseconds, the Juniper QFX5100 fits well in a high-frequency trading architecture.

Private Cloud
: Although the Juniper QFX5100 was specifically designed to solve the challenges of cloud computing and public clouds, you can take advantage of the same features to solve the needs of the private cloud. Enterprises, government agencies, and research institutes are building out their own private clouds, and the Juniper QFX5100 meets and exceeds all their requirements.

Campus
: High port density and a single point of management make the Juniper QFX5100 a perfect fit in a campus architecture, specifically in the core and aggregation roles.

Enterprise
: Offering the flexibility to be used in multiple deployment scenarios, the Juniper QFX5100 gives an enterprise the freedom to use the technology that best fits its needs. The Juniper QFX5100 can be used as a standalone device, a Virtual Chassis, a QFabric Node, a Virtual Chassis Fabric, a MC-LAG, or in a Clos architecture.

It's a very exciting time in the networking industry as SDN, cloud computing, and data center technologies are continuing to push the envelope and bring new innovations and solutions to the field. The Juniper QFX5100 is embracing all of the change

that's happening and providing clear and distinctive solution differentiation. With its wide variety of features, the Juniper QFX5100 is able to quickly solve the challenges of cloud computing as well as other use cases such as high-frequency trading and campus.

Junos

Junos is a purpose-built networking operating system based on one of the most stable and secure operating systems in the world: FreeBSD. Junos is designed as a *monolithic kernel architecture* that places all of the operating system services in the kernel space. Major components of Junos are written as daemons that provide complete process and memory separation.

One of the benefits of monolithic kernel architecture is that kernel functions are executed in supervisor mode on the CPU, whereas the applications and daemons are executed in user space. A single failing daemon will not crash the operating system or impact other unrelated daemons. For example, if there were an issue with the Simple Network Management Protocol (SNMP) daemon and it crashed, it wouldn't impact the routing daemon that handles Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP).

One Junos

Creating a single network operating system that you can use across routers, switches, and firewalls simplifies network operations, administration, and maintenance. Network operators need only learn Junos once and become instantly effective across other Juniper products. An added benefit of a single Junos is that there's no need to reinvent the wheel and have 10 different implementations of BGP or OSPF. Being able to write these core protocols once and then reuse them across all products provides a high level of stability because the code is very mature and field tested.

Software Releases

Every quarter for more than 15 years, there has been a consistent and predictable release of Junos. The development of the core operating system is a single-release train. This allows developers to create new features or fix bugs once and then share them across multiple platforms.

The release numbers are in a major and minor format. The major number is the version of Junos for a particular calendar year, and the minor release indicates in which trimester the software was released. There are a couple of different types of Junos that are released more frequently to resolve issues: maintenance and service releases. Maintenance releases are released about every six weeks to fix a collection of issues, and they are prefixed with "R." For example, Junos 14.1R2 would be the second main-

tenance release for Junos 14.1. Service releases are released on demand to specifically fix a critical issue that has yet to be addressed by a maintenance release. These releases are prefixed with a "S." An example would be Junos 14.1S2.

The general rule of thumb is that new features are added for every major and minor releases and bug fixes are added to service and maintenance releases. For example, Junos 14.1 to 14.2 would introduce new features, whereas Junos 14.1R1 to 14.1R2 would introduce bug fixes.

Most production networks prefer to use the last Junos release of the previous calendar year; these Junos releases are Extended End of Life (EEOL) releases that are supported for three years. The advantage is that the EEOL releases become more stable with time. Consider that 14.1 will stop providing bug fixes after 24 months, whereas 14.3 will continue to include bug fixes for 36 months.

Three-Release Cadence

In 2012, Junos created a new release model to move from four releases per year to three (Table 1-1 and Figure 1-4). This increased the frequency of maintenance releases to resolve more issues more often. The other benefit is that all Junos releases as of 2012 are supported for 24 months, whereas the last release of Junos for the calendar year will still be considered EEOL and have support for 36 months.

Table 1-1. Junos end-of-engineering and end-of-life schedule

Release	Target	End of engineering	End of life
Junos 15.1	March	24 months	+ 6 months
Junos 15.2	July	24 months	+ 6 months
Junos 15.3	November	36 months	+ 6 months

By extending the engineering support and reducing the number of releases, network operators should be able to reduce the frequency of having to upgrade to a new release of code.

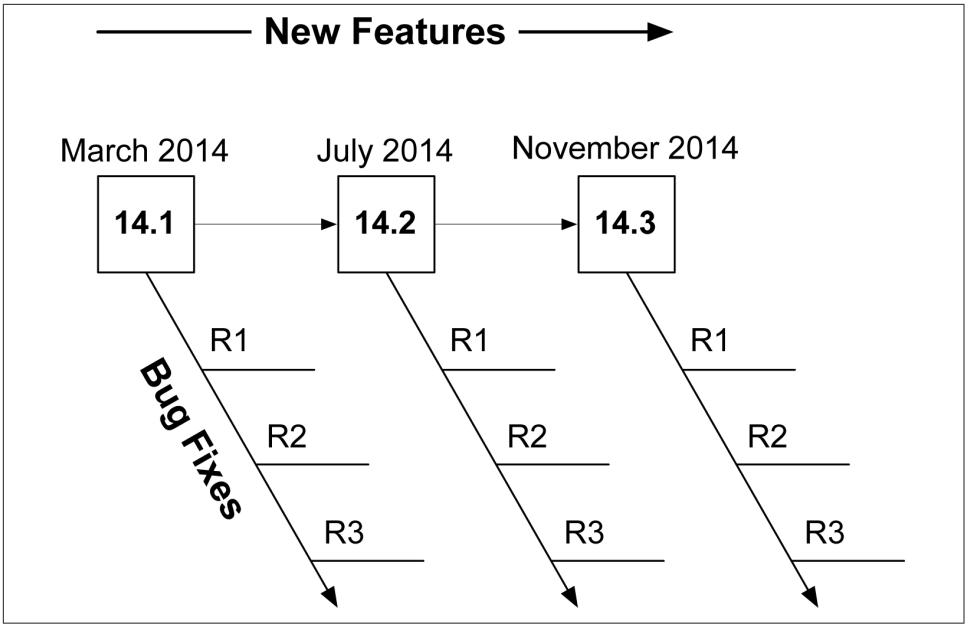


Figure 1-4. Junos three-release cadence

With the new Junos three-release cadence, network operators can be more confident using any version of Junos without feeling pressured to only use the EEOL release.

Software Architecture

Junos was designed from the beginning to support a separation of control and forwarding plane. This is true of the Juniper QFX5100 series for which all of the control plane functions are performed by the routing engine, whereas all of the forwarding is performed by the Packet Forwarding Engine (PFE) (Figure 1-5). Providing this level of separation ensures that one plane doesn't impact the other. For example, the forwarding plane could be forwarding traffic at line rate and performing many different services while the routing engine sits idle and unaffected.

Control plane functions come in many shapes and sizes. There's a common misconception that the control plane only handles routing protocol updates. In fact, there are many more control plane functions. Some examples include:

- Updating the routing table
- Answering SNMP queries
- Processing SSH or HTTP traffic to administer the switch
- Changing fan speed

- Controlling the craft interface
- Providing a Junos micro kernel to the PFEs
- Updating the forwarding table on the PFEs

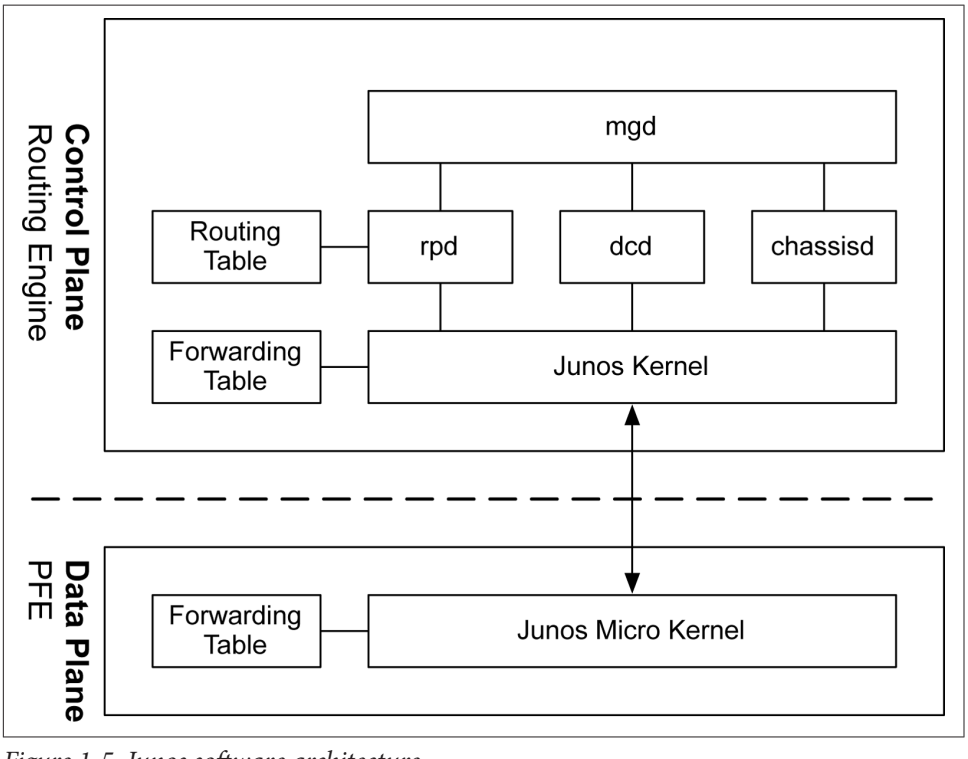


Figure 1-5. Junos software architecture

At a high level, the control plane is implemented entirely within the routing engine, whereas the forwarding plane is implemented within each PFE using a small, purpose-built kernel that contains only the required functions to forward traffic.

The benefit of control and forwarding separation is that any traffic that is being forwarded through the switch will always be processed at line rate on the PFEs and switch fabric; for example, if a switch were processing traffic between web servers and the Internet, all of the processing would be performed by the forwarding plane.

Daemons

The Junos kernel has five major daemons. Each of these daemons play a critical role within the Juniper QFX5100 and work together via Interprocess Communication (IPC) and routing sockets to communicate with the Junos kernel and other daemons.

These daemons, which take center stage and are required for the operation of Junos, are listed here:

- Management daemon (mgd)
- Routing protocol daemon (rpd)
- Device control daemon (dcd)
- Chassis daemon (chassisd)
- Analytics daemon (analyticsd)

There are many more daemons for tasks such as NTP, VRRP, DHCP, and other technologies, but they play a smaller and more specific role in the software architecture. The sections that follow provide descriptions of each of the five major daemons.

Management daemon

The Junos User Interface (UI) keeps everything in a centralized database. This makes it possible for Junos to handle data in interesting ways and opens the door to advanced features such as configuration rollback, apply groups, and activating and deactivating entire portions of the configuration.

The UI has four major components: the configuration database, database schema, management daemon, and the command-line interface (CLI).

The management daemon is the glue that holds the entire Junos UI together. At a high level, it provides a mechanism to process information for both network operators and daemons.

The interactive component of the management daemon is the Junos CLI. This is a terminal-based application that provides the network operator with an interface into Junos. The other side of the management daemon is the XML remote procedure call (RPC) interface. This provides an API through Junoscript and Netconf to accommodate the development of automation applications.

Following are the `cli` responsibilities:

- Command-line editing
- Terminal emulation
- Terminal paging
- Displaying command and variable completions
- Monitoring log files and interfaces
- Executing child processes such as ping, traceroute, and ssh

The management daemon responsibilities include the following:

- Passing commands from the `cli` to the appropriate daemon
- Finding command and variable completions
- Parsing commands

It's interesting to note that the majority of the Junos operational commands use XML to pass data. To see an example of this, simply add the pipe command `display xml` to any command. Let's take a look at a simple command such as `show isis adjacency`:

```
{master}
dhanks@R1-RE0> show isis adjacency
Interface             System         L State       Hold (secs) SNPA
ae0.1                 R2-RE0         2  Up                  23
```

So far, everything looks normal. Let's add the `display xml` to take a closer look:

```
{master}dhanks@R1-RE0> show isis adjacency | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/11.4R1/junos">
    <isis-adjacency-information xmlns="http://xml.juniper.net/junos/11.4R1/junos-
routing" junos:style="brief">
        <isis-adjacency>
            <interface-name>ae0.1</interface-name>
            <system-name>R2-RE0</system-name>
            <level>2</level>
            <adjacency-state>Up</adjacency-state>
            <holdtime>22</holdtime>
        </isis-adjacency>
    </isis-adjacency-information>
   <cli>
        <banner>{master}</banner>
    </cli>
</rpc-reply>
```

As you can see, the data is formatted in XML and received from the management daemon via RPC.

Routing protocol daemon

The routing protocol daemon handles all of the routing protocols configured within Junos. At a high level, its responsibilities are receiving routing advertisements and updates, maintaining the routing table, and installing active routes into the forwarding table. To maintain process separation, each routing protocol configured on the system runs as a separate task within the routing protocol daemon. Its other responsibility is to exchange information with the Junos kernel to receive interface modifications, send route information, and send interface changes.

Let's take a peek into the routing protocol daemon and see what's going on. The hidden command `set task accounting` toggles CPU accounting on and off. Use `show task accounting` to see the results:

```
{master}
dhanks@R1-RE0> set task accounting on
Task accounting enabled.
```

Now, we're good to go. Junos is currently profiling daemons and tasks to get a better idea of what's using the CPU. Let's wait a few minutes for it to collect some data.

OK, let's check it out:

```
{master}
dhanks@R1-RE0> show task accounting
Task accounting is enabled.

Task                        Started    User Time  System Time  Longest Run
Scheduler                       265        0.003        0.000        0.000
Memory                            2        0.000        0.000        0.000
hakr                              1        0.000            0        0.000
ES-IS I/O./var/run/ppmd_c         6        0.000            0        0.000
IS-IS I/O./var/run/ppmd_c        46        0.000        0.000        0.000
PIM I/O./var/run/ppmd_con         9        0.000        0.000        0.000
IS-IS                            90        0.001        0.000        0.000
BFD I/O./var/run/bfdd_con         9        0.000            0        0.000
Mirror Task.128.0.0.6+598        33        0.000        0.000        0.000
KRT                              25        0.000        0.000        0.000
Redirect                          1        0.000        0.000        0.000
MGMT_Listen./var/run/rpd_         7        0.000        0.000        0.000
SNMP Subagent./var/run/sn        15        0.000        0.000        0.000
```

There's not too much going on here, but you get the idea. Currently, running daemons and tasks within the routing protocol daemon are present and accounted for.

The `set task accounting` command is hidden for a reason. It's possible to put additional load on the Junos kernel while accounting is turned on. It isn't recommended to run this command on a production network unless instructed by the Juniper Technical Assistance Center (JTAC). After your debugging is finished, don't forget to turn it back off by using `set task accounting off`.

```
{master}
dhanks@R1-RE0> set task accounting off
Task accounting disabled.
```

Device control daemon

The device control daemon is responsible for setting up interfaces based on the current configuration and available hardware. One feature of Junos is the ability to configure nonexistent hardware. This is based on the underlying assumption that the hardware can be added at a later date and "just work." For example, you can configure `set interfaces ge-1/0/0.0 family inet address 192.168.1.1/24` and commit. Assuming there's no hardware in FPC1, this configuration will not do anything. However, as soon as hardware is installed into FPC1, the first port will be configured immediately with the address 192.168.1.1/24.

Chassis daemon (and friends)

The chassis daemon supports all chassis, alarm, and environmental processes. At a high level, this includes monitoring the health of hardware, managing a real-time database of hardware inventory, and coordinating with the alarm daemon and the craft daemon to manage alarms and LEDs.

It should all seem self-explanatory except for the craft daemon, the craft interface that is the front panel of the device. Let's take a closer look at the Juniper QFX5100 craft interface in Figure 1-6.

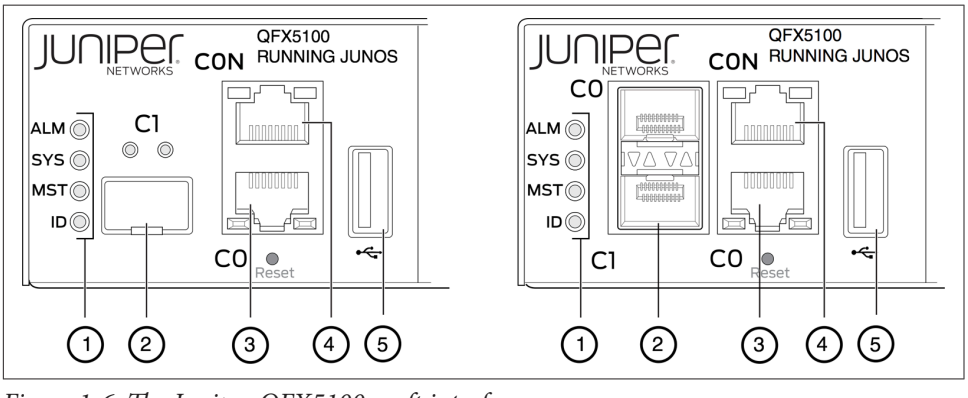


Figure 1-6. The Juniper QFX5100 craft interface

It's simply a collection of buttons and LED lights to display the current status of the hardware and alarms. Let's inspect the LEDs shown in Figure 1-6 a bit closer.

1. Status LEDs.
2. em1-SFP management Ethernet port (C1) cage. It can support either 1GbE copper SFP or fiber SFP.
3. em0-RJ-45 (1000BASE-T) management Ethernet port (C0).
4. RJ-45/RS-232 console port (CON).
5. USB port.

This information can also be obtained via the command line as well with the command `show chassis led`, as illustrated here:

```
{master:0}
dhanks@QFX5100> show chassis led
                LED status for: FPC 0
                ----------------------------------

LEDs status:
    Alarm LED           : Yellow
    System LED          : Green
    Master LED          : Green
    Beacon LED          : Off
```

```
    QIC 1 STATUS LED    : Green
    QIC 2 STATUS LED    : Green

Interface               STATUS LED      LINK/ACTIVITY LED
---------------------------------------------------------
et-0/0/0                N/A             Green
et-0/0/1                N/A             Green
et-0/0/2                N/A             Green
et-0/0/3                N/A             Off
et-0/0/4                N/A             Green
et-0/0/5                N/A             Green
et-0/0/8                N/A             Off
et-0/0/9                N/A             Off
et-0/0/11               N/A             Off
```

One final responsibility of the chassis daemon is monitoring the power and cooling environmentals. It constantly monitors the voltages of all components within the chassis and will send alerts if any of those voltages cross specified thresholds. The same is true for the cooling. The chassis daemon constantly monitors the temperature on all of the components and chips as well as fan speeds. If anything is out of the ordinary, it will create alerts. Under extreme temperature conditions, the chassis daemon can also shut down components to avoid damage.

Analytics daemon

When troubleshooting a network, it's common to ask yourself the following questions:

- Why isn't the application behaving as expected?
- Why is the network slow?
- Am I meeting my service level agreements?

To help answer these questions, the Juniper QFX5100 brings a new daemon into the mix: the analytics daemon. The analytics daemon provides detailed data and reporting on the network's behavior and performance. The data collected can be broken down into two types:

Queue Statistics
: Each port on the switch has the ability to queue data before it is transmitted. The ability to queue data not only ensures the delivery of traffic, but it also impacts the end-to-end latency. The analytics daemon reports data on the queue latency and queue depth at a configured time interval on a per-interface basis.

Traffic Statistics
: Being able to measure the packets per second (pps), packets dropped, port utilization, and number of errors on a per-interface basis gives you the ability to quickly graph the network.

You can access the data collected by the analytics daemon in several different ways. You can store it on the local device or stream it to a remote server in several different formats.

Traffic must be collected from two locations within the switch in order for the data to be accurate. The first location to collect traffic is inside the analytics module in the PFE; this permits the most accurate statistics possible without impacting the switch's performance. The second location to collect traffic is from the routing engine. The PFE sends data to the routing engine if that data exceeds certain thresholds. The analytics daemon will then aggregate the data. The precise statistics directly from the PFE and the aggregated data from the routing engine is combined to give you a complete, end-to-end view of the queue and traffic statistics of the network.

Network analytics is covered in more depth in Chapter 9.

Routing Sockets

Routing sockets are a UNIX mechanism for controlling the routing table. The Junos kernel takes this same mechanism and extends it to include additional information to support additional attributes to create a carrier-class network operating system.

At a high level, there are two actors that use routing sockets: the *state producer* and the *state consumer*. The routing protocol daemon is responsible for processing routing updates and thus is the state producer. Other daemons are considered a state consumer because they process information received from the routing sockets, as demonstrated in Figure 1-7.

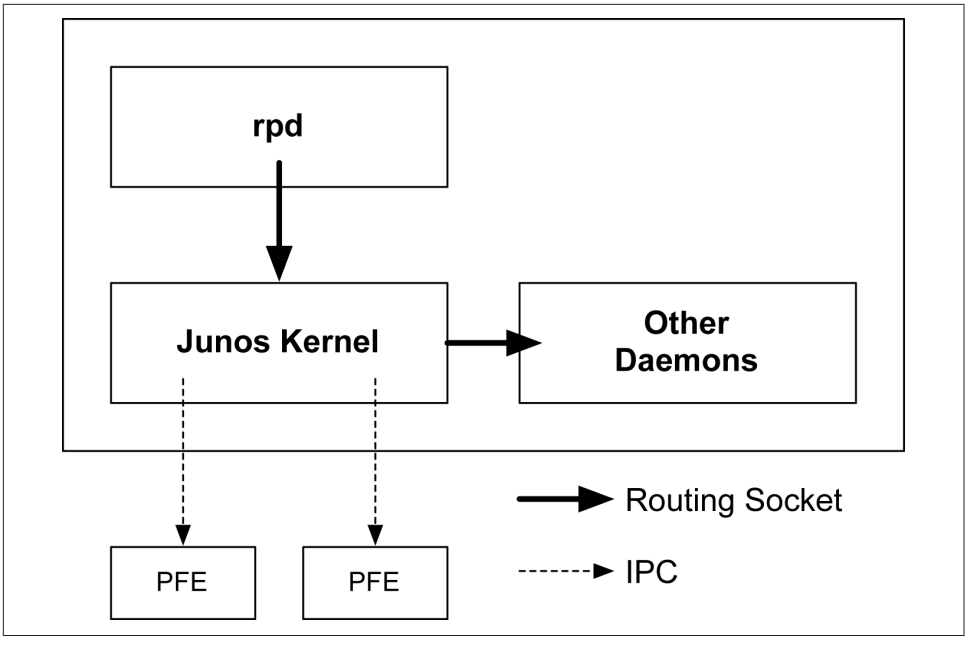


Figure 1-7. Routing socket architecture

Let's take a peek into the routing sockets and see what happens when we configure `ge-1/0/0.0` with an IP address of 192.168.1.1/24. Using the `rtsockmon` command from the shell allows us to see the commands being pushed to the kernel from the Junos daemons.

```
{master}
dhanks@R1-RE0> start shell
dhanks@R1-RE0% rtsockmon -st
        sender    flag   type       op
[16:37:52] dcd      P    iflogical  add     ge-1/0/0.0 flags=0x8000
[16:37:52] dcd      P    ifdev      change  ge-1/0/0 mtu=1514 dflags=0x3
[16:37:52] dcd      P    iffamily   add     inet mtu=1500 flags=0x8000000200000000
[16:37:52] dcd      P    nexthop    add     inet 192.168.1.255 nh=bcst
[16:37:52] dcd      P    nexthop    add     inet 192.168.1.0 nh=recv
[16:37:52] dcd      P    route      add     inet 192.168.1.255
[16:37:52] dcd      P    route      add     inet 192.168.1.0
[16:37:52] dcd      P    route      add     inet 192.168.1.1
[16:37:52] dcd      P    nexthop    add     inet 192.168.1.1 nh=locl
[16:37:52] dcd      P    ifaddr     add     inet local=192.168.1.1
[16:37:52] dcd      P    route      add     inet 192.168.1.1 tid=0
[16:37:52] dcd      P    nexthop    add     inet nh=rslv flags=0x0
[16:37:52] dcd      P    route      add     inet 192.168.1.0 tid=0
[16:37:52] dcd      P    nexthop    change  inet nh=rslv
[16:37:52] dcd      P    ifaddr     add     inet local=192.168.1.1 dest=192.168.1.0
[16:37:52] rpd      P    ifdest     change  ge-1/0/0.0, af 2, up, pfx 192.168.1.0/24
```

For the preceding example, I configured the interface `ge-1/0/0` in a different terminal window and committed the change while the `rtstockmon` command was running.

The command `rtsockmon` is a Junos shell command that gives the user visibility into the messages being passed by the routing socket. The routing sockets are broken into four major components: sender, type, operation, and arguments. The sender field is used to identify which daemon is writing into the routing socket. The type identifies which attribute is being modified. The operation is showing what is actually being performed. There are three basic operations: add, change, and delete. The last field is the arguments passed to the Junos kernel. These are sets of key/value pairs that are being changed.

In the previous example, you can see how `dcd` interacts with the routing socket to configure `ge-1/0/0.0` and assign an IPv4 address.

- `dcd` creates a new logical interface (IFL).
- `dcd` changes the interface device (IFD) to set the proper Maximum Transmission Unit (MTU).
- `dcd` adds a new interface family (IFF) to support IPv4.
- `dcd` sets the nexthop, broadcast, and other attributes that are needed for the Routing Information Base (RIB) and Forwarding Information Base (FIB).
- `dcd` adds the interface address (IFA) of 192.168.1.1.
- `rpd` finally adds a route for 192.168.1.1 and brings it up.

The `rtsockmon` command is used only to demonstrate the functionality of routing sockets and how daemons such as `dcd` and `rpd` use routing sockets to communicate routing changes to the Junos kernel.

QFX5100 Platforms

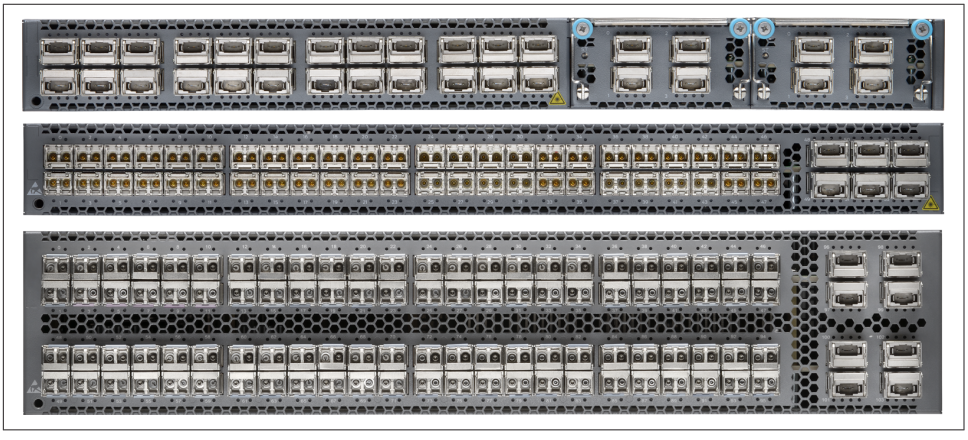


Figure 1-8. The Juniper QFX5100 series

The Juniper QFX5100 series (Figure 1-8) is available in four models. Each has varying numbers of ports and modules, but they share all of the same architecture and benefits. Depending on the number of ports, modules, and use case, a particular model can fit into multiple roles of a data center or campus architecture. In fact, it's common to see the same model of switch in multiple roles in an architecture. For example, if you require 40GbE access, you can use the Juniper QFX5100-24Q in all three roles: core, aggregation, and access. Let's take a look at each model and see how they compare to one another:

QFX5100-24Q
: First is the Juniper QFX5100-24Q; this model has 24 40GbE interfaces and two modules that allow for expansion. In a spine-and-leaf architecture, this model is most commonly deployed as a spine fulfilling the core and aggregation roles.

QFX5100-48S
: Next is the Juniper QFX5100-48S; this model has 48 10GbE interfaces as well as 6 40GbE interfaces. There are no modules, but there is enough bandwidth to provide 2:1 over-subscription. There is 480 Gbps of downstream bandwidth from the 48 10GbE interfaces and 240 Gbps of upstream bandwidth from the 6 40bGE interfaces. In a spine-and-leaf architecture, this model is most commonly deployed as a leaf fulfilling the access role.

QFX5100-96S
: When you need to go big, the Juniper QFX5100-96S offers 96 10GbE and 8 40GbE interfaces to maintain an optimal 3:1 over-subscription. In a spine-and-lead architecture, this model is most commonly deployed as a leaf.

Now that you have an idea of the different models that are available in the Juniper QFX5100 series, let's compare and contrast them in the matrix shown in Table 1-2 so that you can easily see the differences.

Table 1-2. QFX5100 series model comparison

Attribute	QFX5100 24Q	QFX5100 48S	QFX5100 48T	QFX5100 96S
10GbE ports	0	48	48	96
40GbE ports	24	6	6	8
Modules	2	0	0	0
Rack units	1	1	1	2

Depending on how many modules and which specific module is used, the port count can change for the models that have expansion ports. For example, the Juniper QFX5100-24Q has 24 40GbE built-in interfaces, but using two modules can increase the total count to 32 40GbE interfaces, with the assumption that each module has 4 40GbE interfaces. Each model has been specifically designed to operate in a particular role in a data center or campus architecture but offer enough flexibility that a single model can operate in multiple roles.

QFX5100 Modules

The modules make it possible for you to customize the Juniper QFX5100 series to suit the needs of the data center or campus. Depending on the port count and speed of the module, each model can easily be moved between roles in a data center architecture. Let's take a look at the modules available as of this writing:

4 40GbE QIC
: Using this module, you can add an additional 160 Gbps of bandwidth via 4 40GbE interfaces. You can use the interfaces as-is or they can be broken out into 16 10GbE interfaces with a breakout cable.

8 10GbE QIC
: This module adds an additional 80 Gbps of bandwidth via 8 10GbE interfaces. It's a great module to use when you need to add a couple more servers into a rack, assuming the built-in switch interfaces are in use. The eight 10GbE QIC also supports data plane encryption on all eight ports with Media Access Control Security (MACsec).

The QFX Interface Card (QIC) allows you to selectively increase the capacity of the Juniper QFX5100 platforms. Generally, you use the 4 40GbE module for adding additional upstream bandwidth on a ToR or simply filling out the 40GbE interfaces in a spine such as the Juniper QFX5100-24Q. You typically use the 8 10GbE module to add additional downstream bandwidth for connecting compute resources.

QFX5100-24Q

The Juniper QFX5100-24Q (see Figure 1-9) is the workhorse of the Juniper QFX5100 family of switches. In a data center architecture, it can fulfill the roles of the core, aggregation, and access. In a spine-and-leaf topology, it's most commonly used as the spine that interconnects all of the leaves, as shown in Figure 1-10.

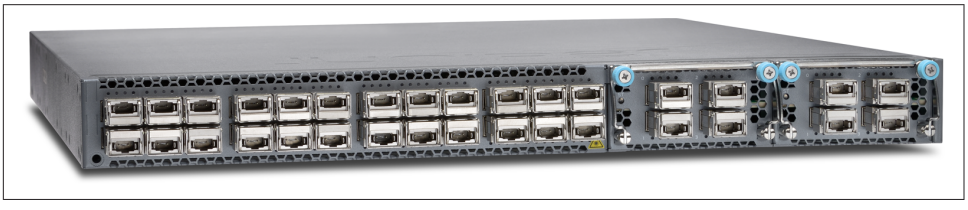


Figure 1-9. The Juniper QFX5100-24Q switch

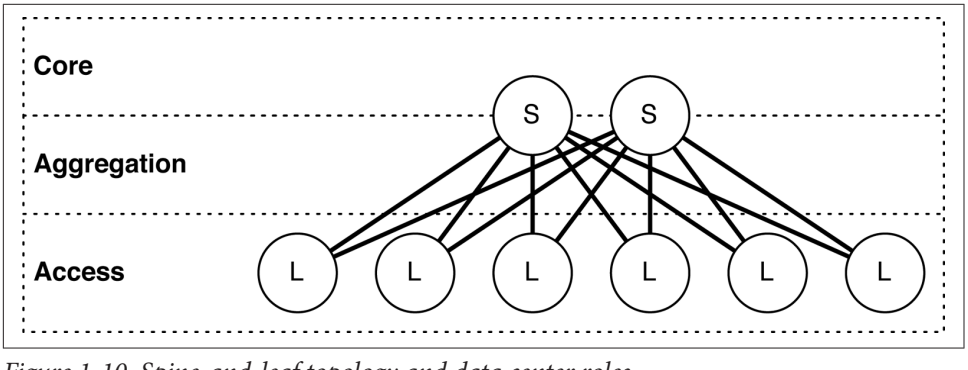


Figure 1-10. Spine-and-leaf topology and data center roles

Roles

An interesting aspect of the Juniper QFX5100-24Q is that it's able to collapse both the core and aggregation roles in a data center architecture. In Figure 1-10, the Juniper QFX5100-24Q is represented by the spine (denoted with an "S"), which is split between the core and aggregation roles. The reason the Juniper QFX5100-24Q is able to collapse the core and aggregation roles is because it offers both high-speed and high-density ports in a single switch.

Module options

The most typical configuration of the Juniper QFX5100-24Q uses a pair of four 40GbE modules, as depicted in Figure 1-11. The combination of built-in ports and modules brings the total interface count to 32 40GbE.

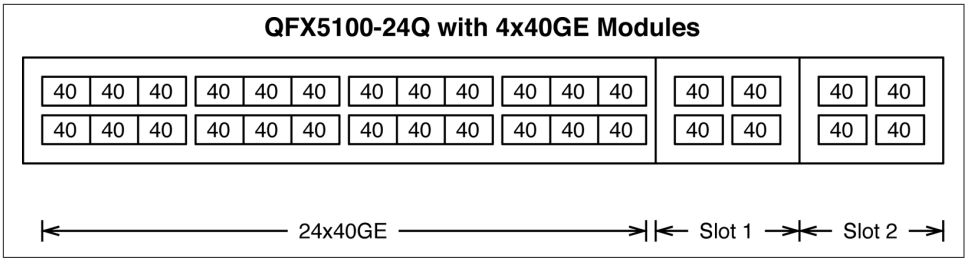


Figure 1-11. The Juniper QFX5100-24Q interface and slot layout using four 40GbE modules

A second configuration using a pair of eight 10GbE QICs can change the Juniper QFX5100-24Q to support 24 40GbE and 16 10GbE interfaces, as illustrated in Figure 1-12.

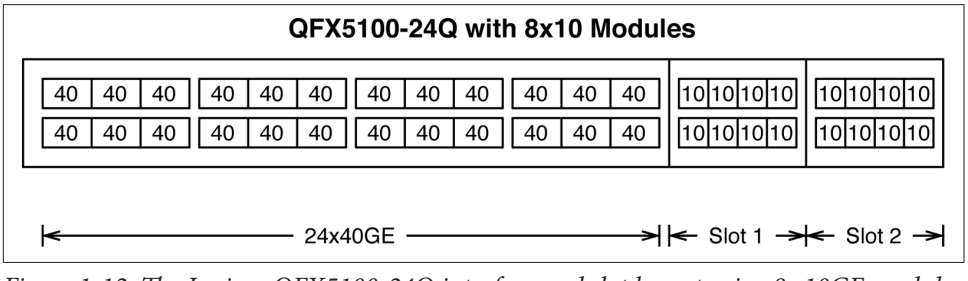


Figure 1-12. The Juniper QFX5100-24Q interface and slot layout using 8×10GE modules

Although the Juniper QFX5100-24Q using the four 40GbE QICs is well suited in the core and aggregation of a spine-and-leaf topology, the eight 10GbE QIC transforms the switch so that it's more suitable as a leaf in the access layer. With a combination of both 10GbE and 40GbE, the Juniper QFX5100-24Q is now able to support a combination of compute resources. In addition, you can still use the switch in the core and aggregation layer in the spine of the topology, but allow other components such as an edge router, firewall, or load balancer to peer directly with the spine, as shown in Figure 1-13.

Let's take a look at Figure 1-13 in more detail to fully understand how you can deploy the Juniper QFX5100-24Q in different roles of an architecture. The spines S1 and S2 are a pair of QFX5100-24Q using the eight 10GbE QICs, which allow the edge routers E1 and E2 to have 10GbE connectivity directly with the spine switches S1 and S2 in the core and aggregation roles. When the Juniper QFX5100-24Q uses the eight 10GbE QICs, it has the flexibility to offer both 10GbE and 40GbE interfaces.

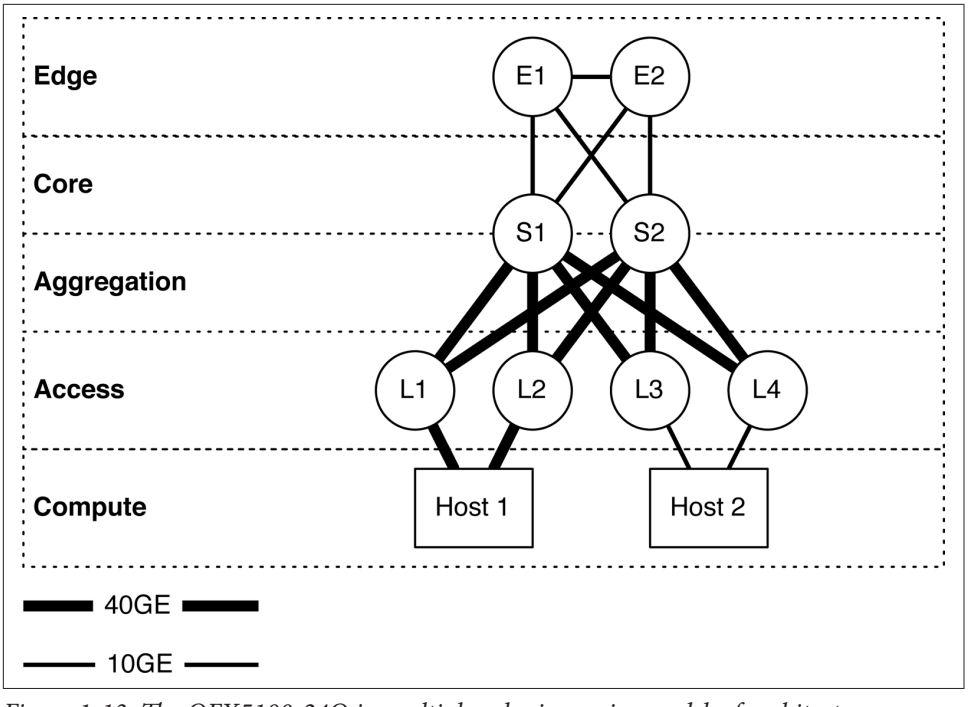


Figure 1-13. The QFX5100-24Q in multiple roles in a spine-and-leaf architecture

In the access role, there are two QFX5100-24Q switches, illustrated as L1 and L2. These switches are providing 40GbE access interfaces to Host 1. The other two access switches, L3 and L4, are a pair of QFX5100-48S switches and provide 10GbE access to Host 2.

Physical attributes

The Juniper QFX5100-24Q is a very flexible switch that you can deploy in a variety of roles in a network. Table 1-3 takes a closer look at the switch's physical attributes.

Table 1-3. Physical attributes of the QFX5100-24Q

Physical attributes	Value
Rack units	1
Built-in interfaces	24 40GbE
Total 10GbE interfaces	104 using breakout cables
Total 40GbE interfaces	32, using two four 40GbE modules
Modules	2
Airflow	Airflow in (AFI) or airflow out (AFO)
Power	150

Physical attributes	Value
Cooling	5 fans with N + 1 redundancy
PFEs	1
Latency	~500 nanoseconds
Buffer size	12 MB

The Juniper QFX5100-24Q packs quite a punch in a small 1RU form factor. As indicated by the model number, the Juniper QFX5100-24Q has 24 40GbE built-in interfaces, but it can support up to 104 10GbE interfaces by using a breakout cable. Although the math says that with 32 40GbE interfaces you should be able to get 128 10GbE interfaces, the PFE has a limitation of 104 total interfaces at any given time. There are two available QIC modules to further expand the switch to support additional 10GbE or 40GbE interfaces.

Cooling is carried out by a set of five fans in a "4 + 1" redundant configuration. You can configure the Juniper QFX5100-24Q to cool front-to-back (AFO) or back-to-front (AFI).

Although the Juniper QFX5100-24Q fans support both AFO and AFI airflow, it's important to match the same airflow with the power supplies. This way, both the fans and power supplies have the same airflow, and the switch is cooled properly. Mismatching the airflow could result in the switch overheating.

The switch is powered by two power supplies. Each power supply can support either AFO or AFI airflow; it's critical that the airflow of the power supply match the airflow of the fans, as shown in Figure 1-14.

Figure 1-14. The rear of the Juniper QFX5100-24Q, illustrating the AFI airflow on the fans and power supplies

A really great feature of the Juniper QFX5100 is the colored plastic on the rear of the switch. The handles to remove the fans and power supplies are color-coded to indicate the direction of airflow.

Blue (AFI)
: Blue represents cool air coming into the rear of the switch, which creates a back-to-front airflow through the chassis.

Orange (AFO)

Orange represents hot air exiting the rear of the switch, which creates a front-to-back airflow through the chassis.

Previously, the AFI and AFO notations were a bit confusing, but with the new color-coding, it's no longer an issue. Being able to quickly identify the type of airflow prevents installation errors and gives you peace of mind.

The rear of the Juniper QFX5100-24Q has three main components, as illustrated in Figure 1-15. These components are management, cooling, and power. The management section (shown on the left in Figure 1-15) has a combination of SFP, 1000BASE-T, RS232, and USB connectivity.

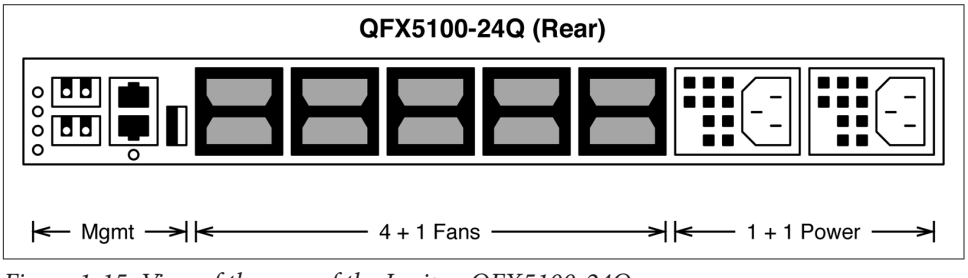


Figure 1-15. View of the rear of the Juniper QFX5100-24Q

There are a total of five fans on the Juniper QFX5100-24, and each one is a field replaceable unit (FRU). The fans are designed in a 4 + 1 redundancy model so that any one of the fans can fail, but the system will continue to operate normally. There is a total of two power supplies operating in a 1 + 1 redundancy configuration. A power supply can experience a failure, and the other power supply has enough output to allow the switch to operate normally.

Management

The management interfaces on the Juniper QFX5100 are very similar to the existing QFX3500 and QFX3600 family. There are six status LEDs, three management ports, an RS-232 port, and a USB port, as illustrated in Figure 1-16.

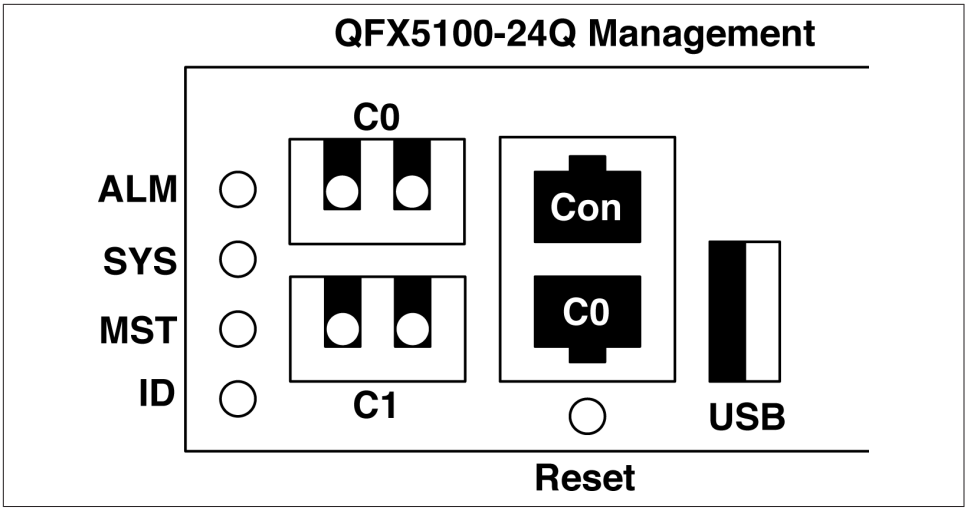


Figure 1-16. The QFX5100-24Q management console

Let's walk through the status LEDs, one by one:

ALM (Alarm)
: The ALM LED can be either red or amber depending on the severity of the alarm. If the alarm is red, this is an indication that one or more hardware components have failed or have exceeded temperature thresholds. An amber alarm indicates a noncritical issue, but if left unchecked, it could result in a service interruption.

SYS (System)
: This LED is always green but has three illumination states: steady, blinking, or off. If the SYS LED is steady and always on, this means that Junos has been properly loaded onto the switch. If the SYS LED is blinking, this means that the switch is still booting. Finally, if the SYS LED is off, it means that the switch is powered off or has been halted.

MST (Master)
: Similar to SYS, the MST LED is always green and has the same three states: steady, blinking, or off. If the MST LED is steady, the switch is currently the master routing engine of a Virtual Chassis. If the MST LED is blinking, the switch is the backup routing engine in a Virtual Chassis. If the MST LED is off, the switch is either a line card in Virtual Chassis or it's operating as a standalone switch.

ID (Identification)
: This is a new LED, first appearing on the Juniper QFX5100 family. It is here to help remote hands and the installation of the switch; you can use it to help identify a particular switch with a visual indicator. The ID LED is always blue and has

two states: on or off. When the ID LED is on, the beacon feature has been enabled through the command line of the switch. If the ID LED is off, this is the default state and indicates that the beacon feature is currently disabled on the switch.

There are three management ports in total, but you can use only two at any given time; these are referred to as C0 and C1. The supported combinations are presented in Table 1-4:

Table 1-4. Valid QFX5100 management port combinations

C0	C1	Transceiver
SFP	SFP	1G-SR, 1G-SR
SFP	SFP	1G-SR, 1G-T
SFP	SFP	1G-T, 1G-SR
SFP	SFP	1G-T, 1G-T
RJ-45	SFP	N/A, 1G-SR
RJ-45	SFP	N/A, 1G-T

Basically, the two C0 management ports are interchangeable, but you can use only one at any given time. The C0 and C1 SFP management port can support either 1G-SR or 1G-T transceivers.

The two management ports C0 and C1 are used for out-of-band management. Typically, only a single management port will be used, but for the scenario in which the Juniper QFX5100 is being used as a QFabric Node, both ports are required, as depicted in Figure 1-17.

In a QFabric architecture, each node and interconnect requires two out-of-band management connections to ensure redundancy. The out-of-band management connections are used purely for the control plane, whereas the 40GbE interfaces are used for the data plane, as illustrated in Figure 1-17. Having both a SFP and copper management port gives you more installation flexibility in the data center. If you prefer fiber, you can easily use just the C1 interface and leave C0 unused. If the switch is being used as a QFabric Node and you require both management ports but only want to use copper, the SFP supports using a 1GE-T transceiver.

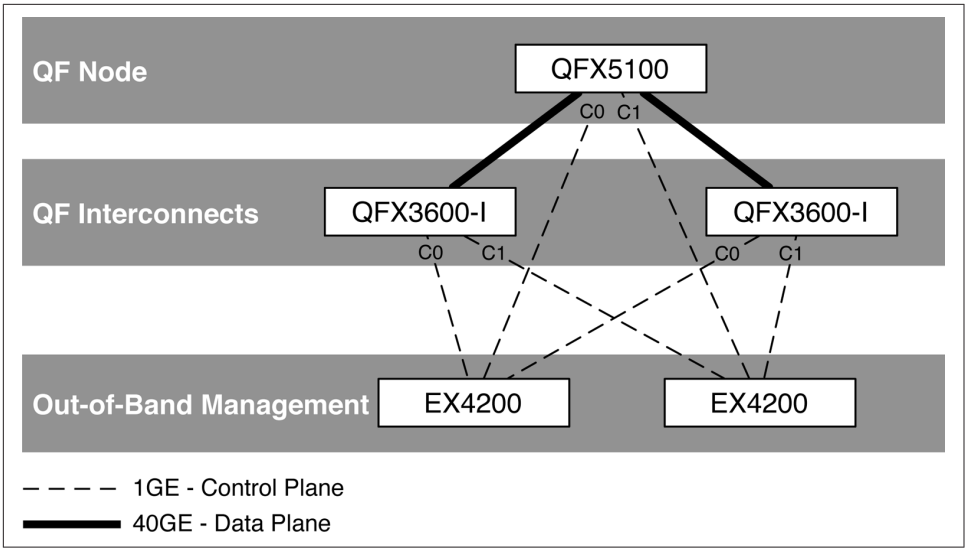


Figure 1-17. The C0 and C1 management ports in a QFabric Node topology

The RS-232 console port is a standard RJ-45 interface. This serial port is used to communicate directly with the routing engine of the switch. For situations in which the switch becomes unreachable by IP, the serial RS-232 is always a nice backup to have.

The USB port is a standard USB 2.0 interface and can be used with any modern thumb drive storage media. Again, for the scenario in which IP connectivity isn't available, you can use the USB port to load software directly onto the switch. The USB port combined with the RS-232 serial console give you full control over the switch.

QFX5100-48S

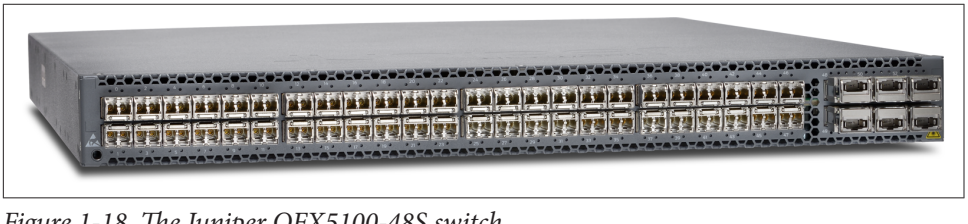


Figure 1-18. The Juniper QFX5100-48S switch

The Juniper QFX5100-48S (Figure 1-18) is another workhorse in the Juniper QFX5100 family of switches. In a data center architecture, it has been designed to fulfill the role of the access tier. In a spine-and-leaf topology, it's most commonly used as the leaf that offers connectivity to end hosts, as shown in Figure 1-19.

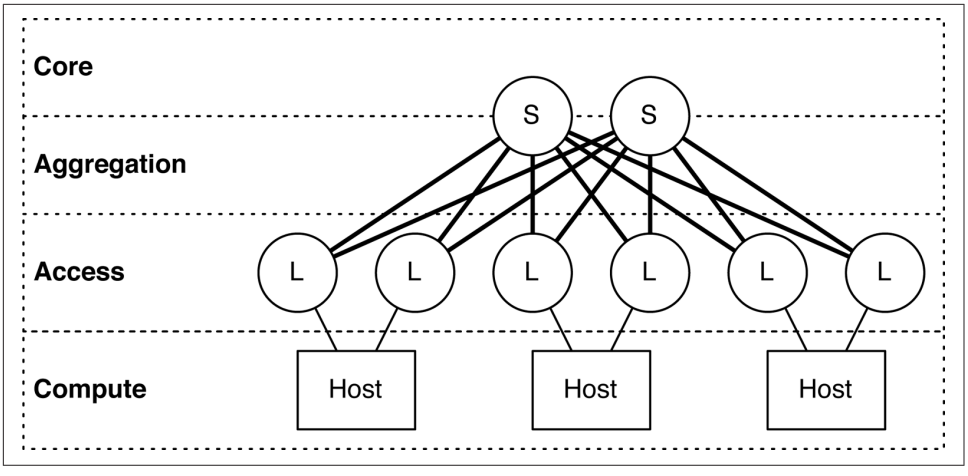


Figure 1-19. Spine-and-leaf topology, with the Juniper QFX5100-48S as a leaf

Roles

The primary role for the Juniper QFX5100-48S is to operate in the access tier of a data center architecture, due to the high density of 10GbE ports. In Figure 1-19, the "L" denotes the Juniper QFX5100-48S in a spine-and-leaf topology; "S" indicates the Juniper QFX5100-24Q being used in the spine. The Juniper QFX5100-24Q and QFX5100-48S were specifically designed to work together to build a spine-and-leaf topology and offer an option of 2:1 or 3:1 over-subscription.

The front of the Juniper QFX5100-48S offers two sets of built-in interfaces: 48 10GbE interfaces and 6 40GbE interfaces, as shown in Figure 1-20. The 48 10GbE interfaces are generally used for end hosts, and the 6 40GbE interfaces are used to connect to the core and aggregation. The 40GbE interfaces can also support 4 10GbE interfaces by using a breakout cable; this brings the total count of 10GbE interfaces to 72 (48 built in + 24 from breakout cables).

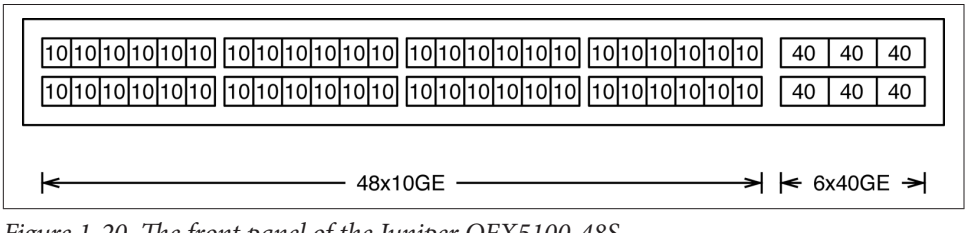


Figure 1-20. The front panel of the Juniper QFX5100-48S

In data centers where the end hosts are only 1GbE, you can change the roles of the Juniper QFX5100-48S and use it as a spine switch in the core and aggregation tiers of a data center architecture. In such a situation, you can use a lower-speed leaf such as

the EX4300 in combination with the Juniper QFX5100-48S to create a spine-and-leaf topology for 1GbE access, as demonstrated in Figure 1-21.

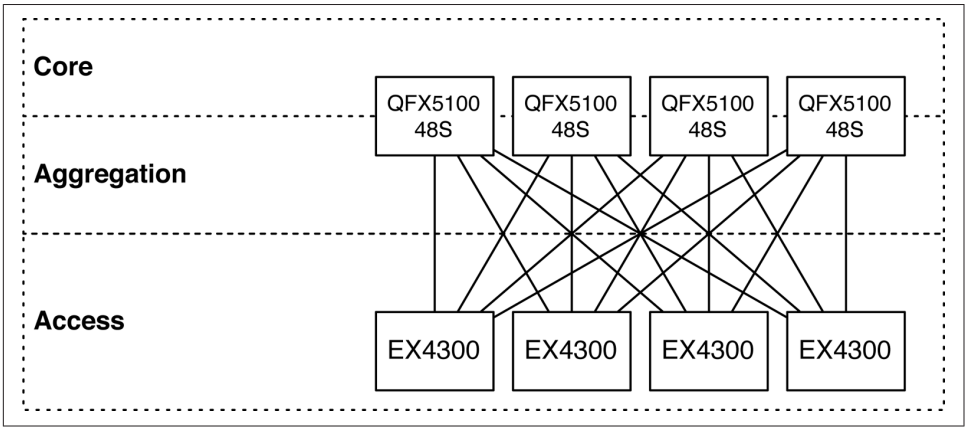


Figure 1-21. Spine-and-leaf topology with the Juniper QFX5100-48S and EX4300

The same logic holds true for a 1GbE spine-and-leaf topology: 1GbE for downstream, and 10GbE for upstream, allowing for an appropriate amount of over-subscription. In the example shown in Figure 1-21, each leaf has 4 10GbE of upstream bandwidth and 48 1GbE of downstream bandwidth; this results in an over-subscription of 1.2:1 which is nearly line rate.

Physical attributes

The Juniper QFX5100-48S is a great access switch. Table 1-5 takes a closer look at the switch's physical attributes.

Table 1-5. Physical attributes of the QFX5100-48S

Physical attributes	Value
Rack units	1
Built-in interfaces	48 10GbE and 6 40GbE
Total 10GbE interfaces	72, using breakout cables
Total 40GbE interfaces	6
Modules	0
Airflow	Airflow in (AFI) or airflow out (AFO)
Power	150
Cooling	5 fans with N + 1 redundancy
PFEs	1
Latency	~500 nanoseconds
Buffer size	12 MB

Aside from the built-in interfaces and modules, the Juniper QFX5100-48S and QFX5100-24 have identical physical attributes. The key to a spine-and-leaf network is that the upstream bandwidth needs to be faster than the downstream bandwidth to ensure an appropriate level of over-subscription.

Management

Just as with the physical attributes, the Juniper QFX5100-48S and QFX5100-24Q are identical in terms of management. The Juniper QFX5100-48S has three management ports, a serial RS-232 port, and a USB port.

QFX5100-48T

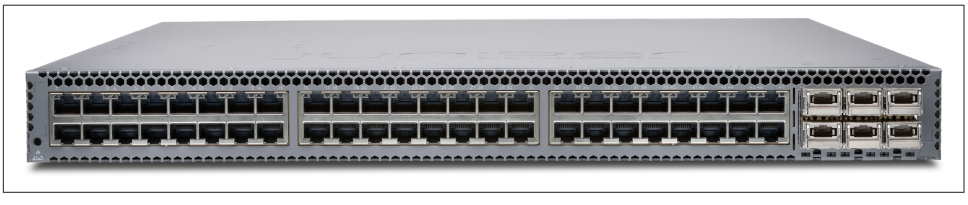


Figure 1-22. The Juniper QFX5100-48T switch

The Juniper QFX5100-48T (Figure 1-22) is very similar to the Juniper QFX5100-48S; the crucial difference is that the Juniper QFX5100-48T supports 10GBASE-T. In a data center architecture, it has been designed to fulfill the role of the access tier. In a spine-and-leaf topology, it's most commonly used as the leaf that offers connectivity to end hosts, as shown in Figure 1-23.

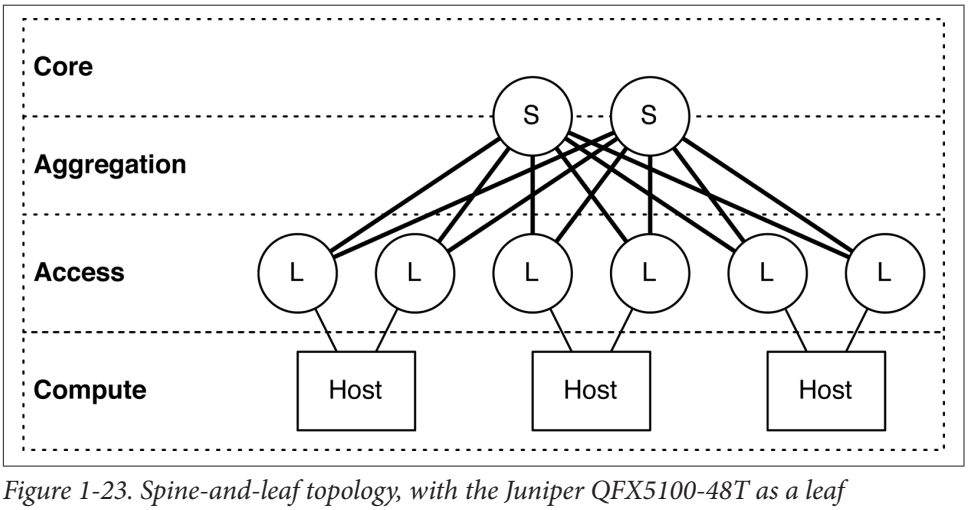


Figure 1-23. Spine-and-leaf topology, with the Juniper QFX5100-48T as a leaf

Roles

The primary role for the Juniper QFX5100-48T is to operate in the access tier of a data center architecture, due to the high density of 10GbE ports. In Figure 1-23, the "L" denotes the Juniper QFX5100-48T in a spine-and-leaf topology; "S" indicates the Juniper QFX5100-24Q being used in the spine. The Juniper QFX5100-24Q and QFX5100-48T were specifically designed to work together to build a spine-and-leaf topology and offer an option of 2:1 or 3:1 over-subscription.

The front of the Juniper QFX5100-48T has two sets of built-in interfaces: 48 10GbE and 6 40GbE, as shown in Figure 1-24. The 48 10GbE interfaces are generally used for end hosts, and the 6 40GbE interfaces are used to connect to the core and aggregation. The 40GbE interfaces can also support 4 10GbE interfaces by using a breakout cable; this brings the total count of 10GbE interfaces to 72 (48 built in + 24 from breakout cables).

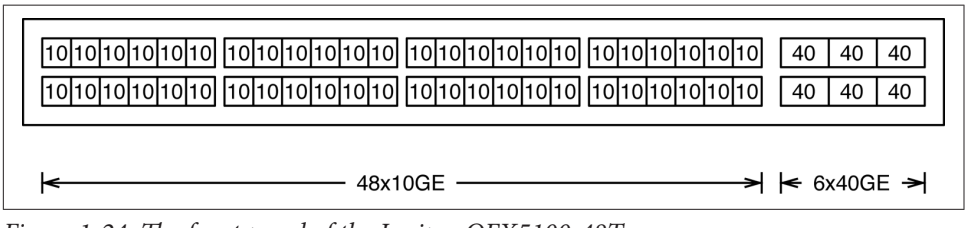


Figure 1-24. The front panel of the Juniper QFX5100-48T

In data centers where the end hosts are only 1GbE, the Juniper QFX5100-48T can support tri-speed interfaces:

- 100 Mbps
- 1 Gbps
- 10 Gbps

The Juniper QFX5100-48T is a very flexible switch in the access layer; network operators can use the same switch for both management and production traffic. Typically, management traffic is 100 Mbps or 1 Gbps over copper by using the RJ-45 interface. New servers just coming to market in 2014 are supporting 10GBASE-T, so the Juniper QFX5100-48T can easily support both slower management traffic as well as blazingly fast production traffic.

Physical attributes

The Juniper QFX5100-48T is a great access switch. Let's take a closer look at the switch's physical attributes in Table 1-6.

Table 1-6. Physical attributes of the QFX5100-48T

Physical Attributes	Value
Rack units	1
Built-in interfaces	48 10GbE and 6 40GbE
Total 10GbE interfaces	48 10GBASE-T and 24 SFP+ using breakout cables
Total 40GbE interfaces	6
Modules	0
Airflow	Airflow in (AFI) or airflow out (AFO)
Power	150
Cooling	5 fans with N + 1 redundancy
PFEs	1
Latency	~500 nanoseconds
Buffer size	12 MB

Aside from the built-in interfaces and modules, the Juniper QFX5100-48T and QFX5100-24Q have identical physical attributes. The key to a spine-and-leaf network is that the upstream bandwidth needs to be faster than the downstream bandwidth to ensure an appropriate level of over-subscription.

Management

The management for the Juniper QFX5100-48T and QFX5100-48S are identical. The Juniper QFX5100-48T has three management ports, a serial RS-232 port, and a USB port.

QFX5100-96S

Go big or go home! The Juniper QFX5100-96S (see Figure 1-25) just happens to be my favorite switch. With 96 10GbE and 8 40GbE ports, it's more than prepared to handle the most dense compute racks. If you don't have enough servers in a rack to make use of this high-density switch, it also makes a great core and aggregation switch.

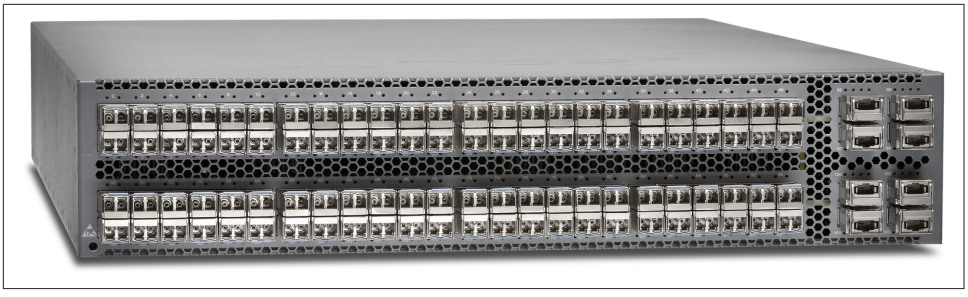


Figure 1-25. The Juniper QFX5100-96S

Roles

The Juniper QFX5100-96S is the king of access switches. As of this writing, it boasts the highest 10GbE port density in a 2RU footprint that Juniper offers. The Juniper QFX5100-96S has 96 10GbE and 8 40GbE built-in interfaces as shown in Figure 1-26.

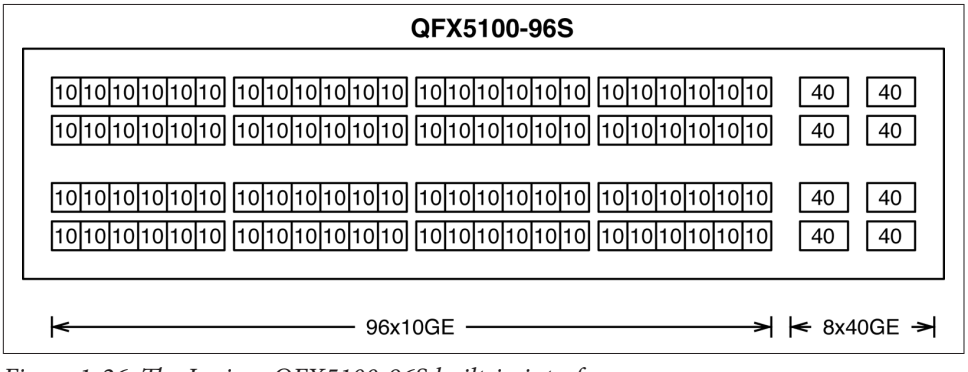


Figure 1-26. The Juniper QFX5100-96S built-in interfaces

The Juniper QFX5100-96S was specifically designed to deliver high-density 10GbE access in the largest data centers in the world, as shown in Figure 1-27; the Juniper QFX5100-96S is in the access tier denoted with an "S."

The other option is to use the Juniper QFX5100-96S as a core and aggregation switch in the spine of the network. Using four QFX5100-96S switches in the spine will offer a dense 384 ports of 10GE.

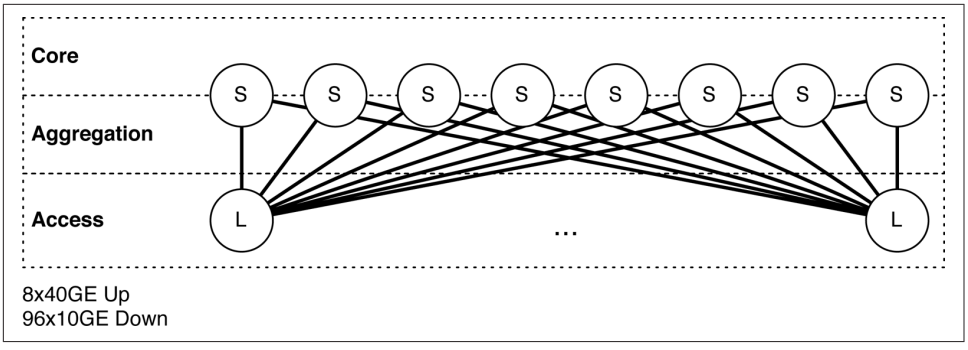


Figure 1-27. The Juniper QFX5100-96S in an access role in a spine-and-leaf topology

Physical attributes

The Juniper QFX5100-96S offers a large amount of 10GbE ports in such a small footprint. Table 1-7 lists the switch's physical attributes.

Table 1-7. Physical attributes of the QFX5100-96S

Physical attributes	Value
Rack units	2
Built-in interfaces	96 10GbE and 8 40GbE
Total 10GbE interfaces	104 using breakout cables on two of the QSFP ports
Total 40GbE interfaces	8
Modules	0
Airflow	Airflow in (AFI) or airflow out (AFO)
Power	150
Cooling	3 fans with N + 1 redundancy
PFEs	1
Latency	~500 nanoseconds
Buffer size	12 MB

Although the Juniper QFX5100-96S can physically support 128 10GbE interfaces, the BRCM 56850 chipset can only support a maximum of 104 logical interfaces.

The Juniper QFX5100-96S was modeled after the Juniper QFX5100-48S; it's basically two QFX5100-48S switches sandwiched together. The Juniper QFX5100-96S pushes the hardware to the limit, offering the maximum amount of performance and total ports. We review the data plane in more detail later in the chapter.

Management

As with the physical attributes, the Juniper QFX5100-96S and QFX5100-48Q are identical in terms of management. The Juniper QFX5100-96S has three management ports, a serial RS-232 port, and a USB port.

Hardware Architecture

The Juniper QFX5100 family shares a lot of the same hardware to keep costs down, reduce the amount of retooling, and increase the overall reliability. The hardware is broken down into the following three major categories:

Chassis
: The chassis houses all of the other components that make up the actual switch. In addition to housing the control plane and data plane, the chassis also controls the environmentals such as power and cooling.

Control Board
: The control board is responsible for many management aspects of the switch. It is essentially a custom motherboard that brings together the control plane CPU, memory, solid-state disks (SSDs), I2C connections, and other management modules. The Juniper QFX5100 family uses Linux and KVM to virtualize the network operating system—Junos—which is responsible for all of the management, routing protocols, and other exception traffic in the switch.

Switch Board
: The switch board brings together the built-in interfaces, expansion modules, application-specific integrated circuit (ASIC), and the precision timing module. All of the heavy lifting in terms of forwarding traffic is always processed by the data plane. Its sole purpose is to forward traffic from port to port as fast as possible.

All three components work together to bring the switch to life and make it possible for it to forward Ethernet frames in a data center. For the switch to function, all three components must be present. Given the critical nature of each component, it's a requirement that redundancy and high availability must be a priority in the design of a switch.

Let's take a look at the overall hardware architecture of the Juniper QFX5100 family. Each model is going to be a little different in terms of interfaces, modules, and rack units, but the major components are all the same, as is demonstrated in Figure 1-28.

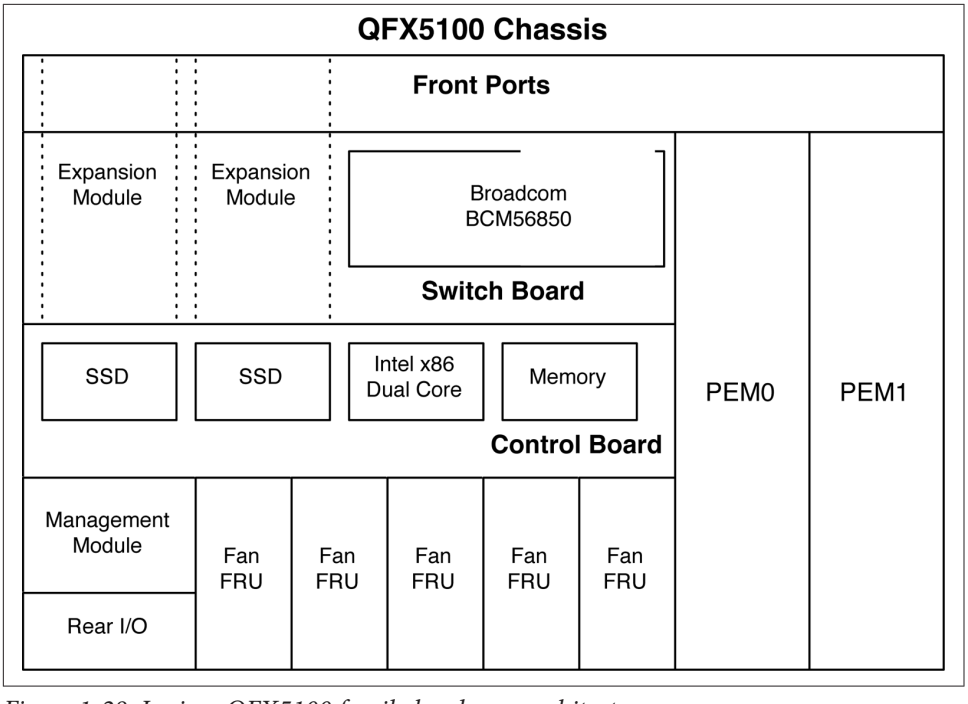


Figure 1-28. Juniper QFX5100 family hardware architecture

The switch board holds together all of the components that make up the data plane; this includes all of the built-in interfaces, modules, and the Broadcom Trident II chipset. The control board houses all of the components needed to run the control plane and manage the chassis and switch board. The two power supplies are labeled as PEM0 and PEM1 (Power Entry Module). The management module is responsible for the two management interfaces, RS232 port, and USB port. Finally, the five fans in the Juniper QFX5100-24Q and QFX5100-48 are aligned in the rear of the switch so that the airflow cools the entire chassis.

Chassis

The chassis, which physically defines the shape and size of the switch, is responsible for bringing everything together. Its most important responsibility is providing power and cooling to all of the other components within it. Let's examine each component to learn a bit more about the chassis.

Power

Each switch in the Juniper QFX5100 family requires two power supplies to support a 1 + 1 redundancy. In the event of a failure, the switch can operate on a single power supply. There are two types of power supplies: airflow in (AFI) and airflow out

(AFO). The fans and power supplies must have the same airflow direction or the chassis will trigger an alarm. Each power supply is color-coded to help quickly identify the airflow direction. AFO is colored orange, and the AFI is colored blue.

Each power supply is 650 W, but the power draw is only about 280 W for a fully loaded system. On the Juniper QFX5100-96S with 96 10GbE interfaces, the average power usage is 2.9 W per 10GbE port. On the Juniper QFX5100-24Q with 32 40GbE interfaces, the average consumption is 8.7 W per 40GbE port.

Cooling

The Juniper QFX5100 family was designed specifically for the data center environment; each system supports front-to-back cooling with reversible airflow. Each chassis has a total of five fans; each fan can be either AFO or AFI. All power supplies and fans must be either AFO or AFI, otherwise the chassis will issue alarms.

Sensors

Each chassis has a minimum of seven temperature sensors, whereas chassis that support modules have a total of nine sensors. This is so each module has its own temperature sensor. Each sensor has a set of configurable thresholds that can raise a warning alarm or shutdown the switch. For example if the CPU were running at 86° C, the switch would sound a warning alarm; however, if the temperate were to rise to 92° C, it would shut down the system to prevent damage.

If you want to see what the current temperatures and fan speeds are, use the `show chassis environment` command, as shown in the following:

```
dhanks@opus> show chassis environment
Class Item                           Status     Measurement
Power FPC 0 Power Supply 0           OK
      FPC 0 Power Supply 1           OK
Temp  FPC 0 Sensor TopMiddle E       OK         29 degrees C / 84 degrees F
      FPC 0 Sensor TopRight I        OK         24 degrees C / 75 degrees F
      FPC 0 Sensor TopLeft I         OK         27 degrees C / 80 degrees F
      FPC 0 Sensor TopRight E        OK         25 degrees C / 77 degrees F
      FPC 0 Sensor CPURight I        OK         30 degrees C / 86 degrees F
      FPC 0 Sensor CPULeft I         OK         28 degrees C / 82 degrees F
      FPC 0 Sensor CPU Die Temp      OK         45 degrees C / 113 degrees F
Fans  FPC 0 Fan Tray 0               OK         Spinning at normal speed
      FPC 0 Fan Tray 1               OK         Spinning at normal speed
      FPC 0 Fan Tray 2               OK         Spinning at normal speed
      FPC 0 Fan Tray 3               OK         Spinning at normal speed
      FPC 0 Fan Tray 4               OK         Spinning at normal speed
```

To view the default thresholds, use the `show chassis temperature-thresholds commands`, as demonstrated here:

```
dhanks@opus> show chassis temperature-thresholds
                            Fan speed      Yellow alarm      Red alarm      Fire Shutdown
                           (degrees C)      (degrees C)     (degrees C)      (degrees C)
Item                      Normal  High   Normal  Bad fan   Normal  Bad fan     Normal
```

```
FPC 3 Sensor TopMiddle E     47    67       65       65       71       71
FPC 3 Sensor TopRight I      41    65       63       63       69       69
FPC 3 Sensor TopLeft I       45    67       64       64       70       70
FPC 3 Sensor TopRight E      42    64       62       62       68       68
FPC 3 Sensor CPURight I      40    67       65       65       71       71
FPC 3 Sensor CPULeft I       44    65       63       63       69       69
FPC 3 Sensor CPU Die Temp    62    93       86       86       92       92
```

It's important that you review the default sensor thresholds and see if they're appropriate for your environment; they're your insurance policy against physically damaging the switch in harsh environments.

Control Plane

The control plane is essentially the brain of the switch. It encompasses a wide variety of responsibilities that can be broken down into the following four categories:

Management
: There are various ways to manage a switch. Some common examples are SSH, Telnet, SNMP, and NETCONF.

Configuration and Provisioning
: There are tools and protocols to change the way the switch operates and modify state. Some examples include Puppet, Chef, Device Management Interface (DMI), Open vSwitch Database (OVSDB), and OpenFlow.

Routing Protocols
: For a switch to participate in a network topology, it's common that the switch needs to run a routing protocol. Some examples include OSPF, IS-IS, and BGP.

Switching Protocols
: The same goes for Layer 2 protocols, such as LLDP, STP, LACP, and MC-LAG.

As described earlier in the chapter, Junos, the network operating system, is responsible for all of the preceding functions.

The Juniper QFX5100 has a little trick up its sleeve. It takes virtualization to heart and uses Linux and KVM to create its own virtualization framework (see Figure 1-30). This creates two immediate benefits:

Two Routing Engines
: Even though the control board has a single dual-core CPU, taking advantage of virtualization, the Juniper QFX5100 is able to have two routing engines. One of the primary benefits of two routing engines is the set of high-availability features. The Juniper QFX5100 is able to take full advantage of Nonstop Routing (NSR), Nonstop Bridging (NSB), Graceful Routing Engine Failover (GRES), and In-Service Software Upgrade (ISSU).

Snapshots
: One of the great aspects of hypervisors is that they can take a snapshot of a virtual machine and have the ability to revert back to a snapshot at any time. Do you have a big upgrade planned? Need a backout plan? Snapshots to the rescue.

Let's take a look at how the Juniper QFX5100 is able to virtualize the control plane. Figure 1-29 shows that the main tool being used is Linux and KVM.

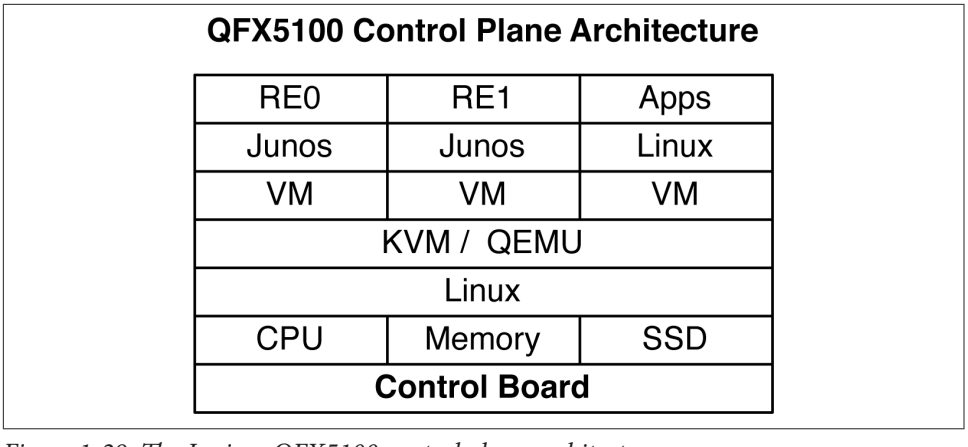


Figure 1-29. The Juniper QFX5100 control plane architecture

The Juniper QFX5100 is able to reap all of the high-availability benefits through virtualization that are usually reserved for high-end systems such as the Juniper MX and T Series. In addition to having two routing engines, there's enough space remaining to have a third virtual machine that you can use for third-party applications. Perhaps you have some management scripts that need to be hosted locally, and you don't want it to interfere with the switch's control plane. No problem, there's a VM for that.

Are you interested in control plane virtualization and want to learn more? Chapter 2 is dedicated to just that and shows you how all of this works and is put together.

Processor, memory, and storage

The Juniper QFX5100 uses a modern Intel dual-core CPU based on the Sandy Bridge architecture. The processor speed is 1.5 GHz, which is more than adequate for two routing engines and third-party applications. The control board has 8 GB of memory. Finally, the control board has a pair of 16 GB high-speed SSDs. Overall the Juniper QFX5100 has a really zippy control plane, and you will enjoy the fast commit times and quickness of the Junos CLI.

Data Plane

So, let's get right to it. The data plane is driven by a Broadcom BCM56850 chipset, which is also known as the Trident II. As of this writing, it's one of the latest 10/40GbE chipsets on the market from Broadcom. The Trident II chipset brings many great features of which you can take advantage:

Single Chipset
: *Switch on a Chip* (SoC) is a concept whereby the entire data plane of a switch is driven by a single chipset. The advantage of this architecture is that it offers much lower latency as compared to multiple chipsets. The Trident II chipset has enough ports and throughput to drive the entire switch.

Overlay Networking
: With the rise of SDN, new protocols such as VXLAN and NVGRE are being used to decouple the network from the physical hardware. The Trident II chipset supports the tunnel termination of both VXLAN and NVGRE in hardware with no performance loss.

Bigger, Better, Faster
: The Trident II chipset has more ports, more bandwidth, and higher throughput; this allows the creation of better switches that can support a wide variety of port configurations. Creating a family of switches that can be deployed in multiple roles within a data center architecture using the same chipset has many advantages for both the customer and vendor.

Merchant silicon

Chipsets such as the Broadcom Trident II are often referred to as *merchant silicon* or "off-the-shelf silicon" (OTS). Many networking vendors offer similar networking switches that are based off the same chipsets as the Trident II. It would be an incorrect assumption that networking switches that are based on the same chipsets are identical in function. Recall from Figure 1-28, that the architecture of a network switch includes three primary components: the chassis, switch board, and control board. The control plane (control board) and the data plane (switch board) must both be programmed and synchronized in order to provide a networking service or feature. In other words, although the chipset might support a specific feature, unless the control plane also supports it, you won't be able to take advantage of it. Given the importance of the control plane, it becomes increasingly critical when comparing different network switches.

Having casual knowledge of the various chipsets helps when you need to quickly access the "speeds and feeds" of a particular network switch. For example, network switches based on the Trident II chipset will generally support 10GbE and 40GbE interfaces, allow up to 104 interfaces, and will not exceed 1,280 Gbps of overall

throughput. Such limitations exist because it's the inherent limitation of the underlying chipset used in the network switch.

It's absolutely required that both the control plane and data plane support a particular network feature or service in order for that service or switch to be usable. Sometimes network vendors state that a switch is "Foobar Enabled" or "Foobar Ready," which merely hints that the chipset itself supports it, but the control plane doesn't and requires additional development. If the vendor is being extra tricky, you will see "Foobar*," with the asterisk denoting that the feature will be released—via the control plane—in the future.

The control plane really brings the network switch to life; it's the brains behind the entire switch. Without the control plane, the switch is just a piece of metal and silicon. This poses an interesting question: if multiple network vendors use the same chipsets in the data plane, how do you choose which one is the best for your network? The answer is simple and has two sides: the first is the switch board flexibility, and the second is the differentiation that comes through the control plane.

Even though the chipset might be the same, vendors aren't limited in how they can use it to create a switch. For example, using the same chipset, you could build a fixed-port network switch or create a network switch that uses modules. Some configurations make sense such as the 48 10GbE, because this is a common footprint for a compute rack. There are other use cases besides an access switch; having the flexibility to use different modules in the same switch allows you to place the same switch in multiple roles in a data center.

More than any other component, the control plane impacts what is and what isn't possible with a network switch that's based off merchant silicon. Some vendors limit the number of data center technologies that are enabled on the switch. For example, you can only use the switch in an Ethernet fabric or it only supports features to build a spine-and-leaf network. Right off the bat, the control plane has already limited where you can and cannot use the switch. What do you do when you want to build an Ethernet fabric, but the switch can only be used in a spine-and-leaf network? The Juniper QFX5100 family is known for "one box, many options" because it offers six different switching technologies to build a network. You can read more about these technologies in Chapter 3.

There are many inherent benefits that come with virtualizing the control plane by using a hypervisor. One is the ability to create snapshots and roll back to a previous known-good state. Another is the ability to have multiple control planes and routing engines to enable features such as ISSU with which you can upgrade the switch without dropping any traffic.

The control plane makes or breaks the switch. It's crucial that you're familiar with the features and capabilities of the networking operating system. Junos has a very strong

pedigree in the networking world and has been developed over the past 15 years, which results in a very stable, robust, and feature-rich control plane. The control plane is so critical that part of this chapter is dedicated to the Junos architecture, and Chapter 2 focuses directly on the control plane virtualization architecture.

Architecture

So, let's consider the Trident II chipset architecture. First and foremost is that the Trident II has enough throughput and supports enough logical ports that it can drive the entire network switch itself. Using a single chipset enables an SoC design that lowers the overall power consumption and port-to-port latency. The Trident II chipset has eight primary engines, as depicted in Figure 1-30.

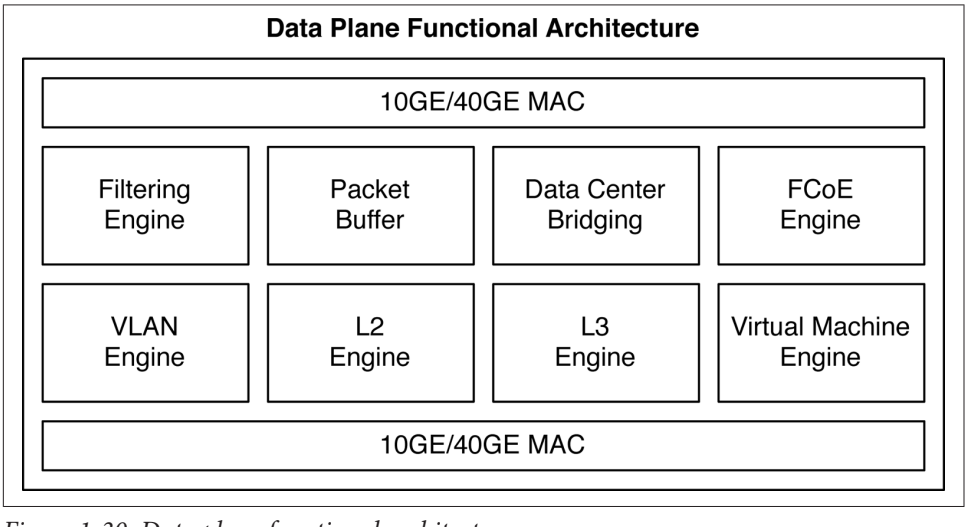


Figure 1-30. Data plane functional architecture

Data is handled by the 10GbE and 40GbE interfaces and is processed by the eight internal traffic engines. Each engine represents a discrete step in processing each Ethernet frame that flows through the switch. By looking at the available functions of a chipset, you can make some immediate assumptions regarding where the chipset can be used in the network. For example, the chipset functions in Figure 1-30 are appropriate for a data center or campus network, but they wouldn't be applicable for an edge and aggregation network or optical-core network. You have to use a chipset with the appropriate functions that match the role in the network; this is why vendors use merchant silicon for some devices and custom silicon for others, depending on the use case.

Life of a frame

As an Ethernet frame makes its way from one port to another, it has to move through different processing engines in the data plane, as shown in Figure 1-31.

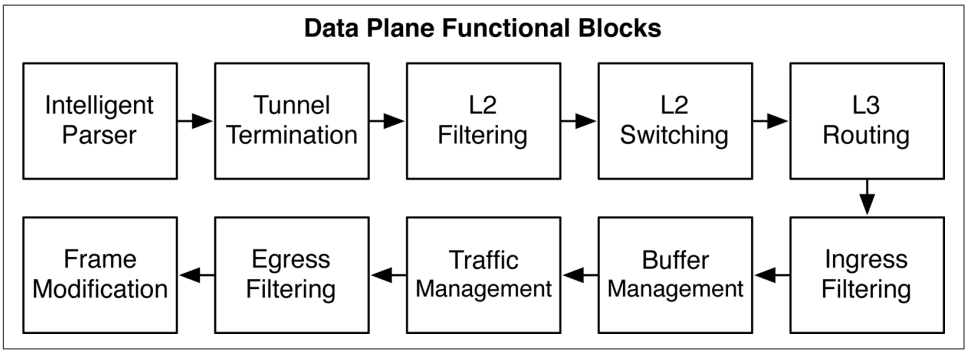


Figure 1-31. Data plane function blocks

Each functional block makes modifications to the Ethernet frame and then passes it to the next functional block in the workflow. Each functional block has a very specific function and role in processing the Ethernet frame. The end result is that as an Ethernet packet flows through the switch, it's able to be manipulated by a wide variety of services without a loss of performance. The following are descriptions of each functional block:

Intelligent Parser
: The first step is to parse the first 128 bytes of the Ethernet frame. Various information, such as the Layer 2 header, Ethernet Type, Layer 3 header, and protocols are saved into memory so that other functional blocks can quickly access this information.

Tunnel Termination
: The next step is to inspect the Ethernet frame in more detail and determine if the switch needs to be a termination point of any tunnel protocols, such as VXLAN, GRE, and MPLS.

Layer 2 Filtering
: This functional block is a preprocessor to determine where to route Layer 2 and Layer 3 packets. During this phase the packet can be moved to a different VLAN or VRF depending on the information in the first 128 bytes.

Layer 2 Switching
: During this stage of the process, the switch needs to process all Layer 2 functions such as VLAN switching, process double-tags, and process encapsulations such as GRE, MPLS, or VXLAN.

Layer 3 Routing
: When all of the Layer 2 processing is complete, the next stage in the process is Layer 3. The Layer 3 routing functional block is responsible for unicast and multicast lookups, longest prefix matching, and unicast reverse path forwarding (uRPF).

Ingress Filtering
: The most powerful filtering happens in the ingress filtering functional block. The filtering happens in two stages: match and action. Nearly any field in the first 128 bytes of the packet can be used to identify and match fields that should be subject to further processing. The actions could be to permit, drop, or change the forwarding class, or assign a new next hop.

Buffer Management
: All Quality of Service features, such as congestion management, classification, queuing, and scheduling are performed in the buffer management functional block.

Traffic Management
: If the Ethernet frame is subject to hashing such as LACP or ECMP, the packet is run through the hashing algorithm to select the appropriate next hop. Support for storm control for broadcast, multicast, and unknown unicast (BUM) is managed by the traffic management functional block.

Egress Filtering
: Sometimes, it's desirable to filter packets on egress. The egress filtering functional block is identical to the ingress filtering functional block, except that the matching and actions are performed only for egress packets.

Frame Modification
: Depending on all of the prior processing of the Ethernet frame, the final egress frame might require substantial modification. In the simplest form the switch can merely just decrement the IP time-to-live (TTL). A more complicated example would be that an Ethernet frame needs to be encapsulated with VXLAN or MPLS.

In summary, there are 10 discrete functional blocks that have specific roles and responsibilities. Starting with intelligent parsing and ending with frame modification, the life of a frame is subject to an end-to-end workload. Separating out the functions into different blocks facilitates predictable behavior and latency as the Ethernet frame makes its way through the switch.

Design Options

Let's take a look at some of the most common design options, and understand what the differences are between them, and what challenges they solve. The following design options will only make use of the Juniper QFX5100 family platforms. There are other design options that can include QFabric and other platforms, but we will review these in detail later in the book.

768×10GbE Ethernet Fabric

The first and most common design option is to create an Ethernet fabric using the Juniper QFX5100 family. The Ethernet fabric—the VCF—is able to provide a single point of management, 3:1 over-subscription, and FCoE support. VCF will be covered in detail in Chapter 6, but let's take a sneak peek at the topology and design benefits.

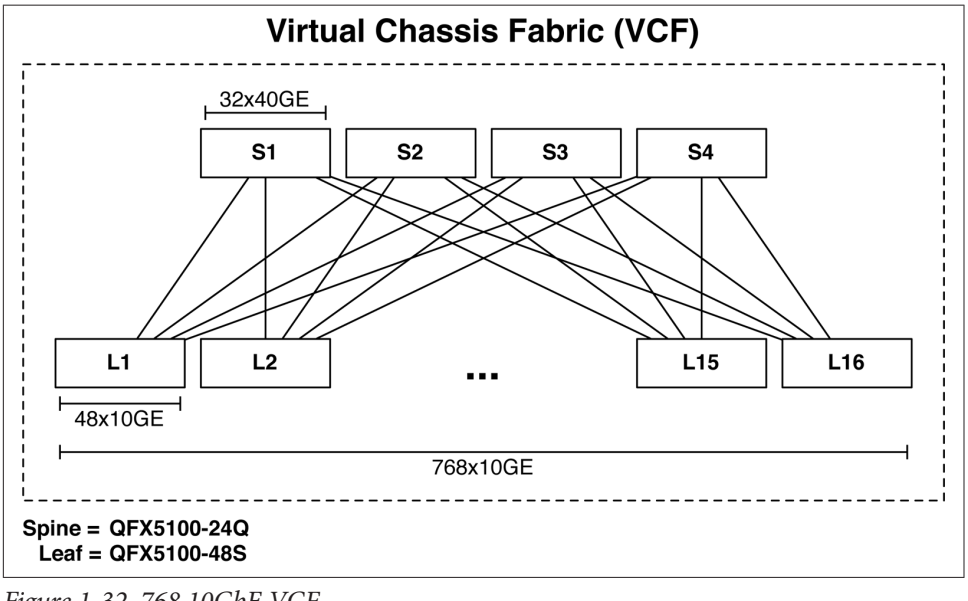


Figure 1-32. 768 10GbE VCF

As of this writing, a VCF has only two rules:

- The number of spines cannot exceed four members.
- The total number of switches in the fabric cannot exceed 20 members.

The example in Figure 1-32 has four spines using the Juniper QFX5100-24Q and 16 leaves using the Juniper QFX5100-48S; the total number of switches in this design option is 20. Each spine has 32 40GbE interfaces, and each leaf has 48 10GbE and 6

40GbE interfaces. Each leaf is using only 4 40GbE interfaces to connect to the spine, so the total over-subscription is 480:160 or 3:1.

The astute reader will realize that only 4 out of the 6 40GbE interfaces on the Juniper QFX5100-48S is being used. In addition, there are only 16 out of 32 40GbE interfaces being used on the Juniper QFX5100-24Q in the spine. This is because of the current limitation of 20 devices in a VCF.

Building a VCF with 768 10GbE interfaces is perfect for many small to medium-sized data centers. Let's explore the benefits of this VCF design option:

- 768 10GbE ports
- Single point of management and control
- Full support for FCoE and converged storage
- Topology-independent ISSU
- Full ECMP for both Layer 2 and Layer 3
- Plug-and-play implementation
- In-band control plane with no additional equipment required
- 1.5 µs end-to-end latency

With the assumption that each server requires two 10GbE links, a VCF network is a great way to easily manage 384 servers with a single Ethernet fabric that supports converged storage and the ability to upgrade the software without traffic loss. Also, assume that each server can support up to 20 VMs; VCF can support 384 servers and 7,680 VMs.

3,072 10GbE Clos

A common alternative to building a Virtual Chassis Fabric is to build a Layer 3 Clos Fabric, as illustrated in Figure 1-33. One of the key benefits of a Clos fabric is the large scale; in this example, the Clos fabric is able to scale up to 3,072 10GbE ports.

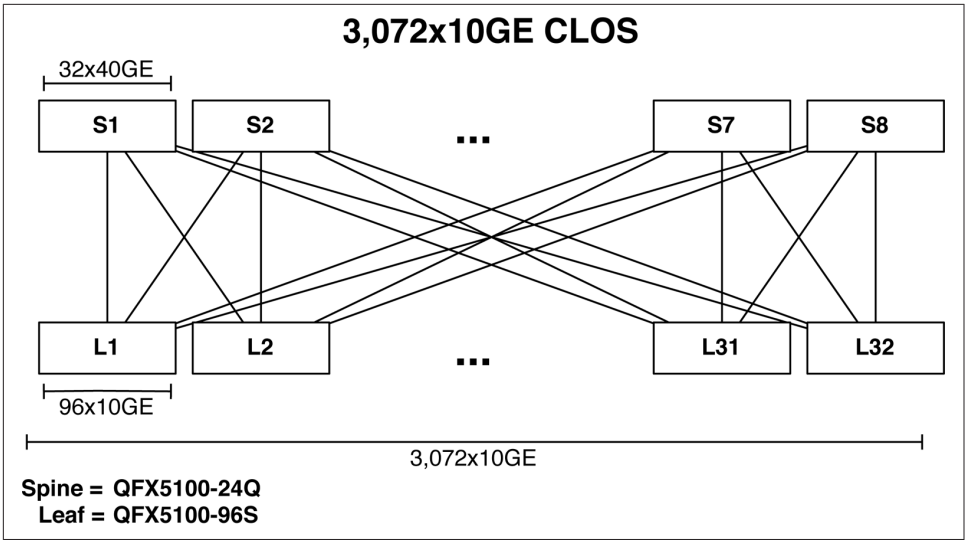


Figure 1-33. 3,072 10GbE Clos fabric

Clos fabrics are covered in greater depth in Chapter 8, but let's take a look at some of the key features of this design option. The speed and number of uplink interfaces of the leaf switch determine the shape and size of the spine. In Figure 1-33, the leaf switch is the Juniper QFX5100-96S, which has 8 40GbE uplinks. To build a simple Clos, the easiest option is to select a spine switch that supports 40GbE interfaces and have eight of them, which matches the number of uplinks on the Juniper QFX5100-96S. In this example, we'll use the Juniper QFX5100-24Q in the spine, which results in a total of 32 40GbE interfaces. With a total of eight spine switches, this results in 256 40GbE interfaces that can be used by the leaf switches. Because each QFX5100-96S has eight uplinks, we'll have a total of 32 leaves (256 ports/8 uplinks = 32 leaves). Each leaf has 96 10GbE interfaces, thus the Clos fabric has 3,072 10GbE interfaces (96 ports × 32 leaves = 3,072 10GbE).

One of the central assumptions with a Layer 3 Clos is that each switch uses a routing protocol such as OSPF, IS-IS, or BGP to connect to one another. If you have a virtualized workload and want to use an overlay architecture, a Layer 3 Clos is the perfect way to scale out your data center.

12,288 10GbE Clos

Now that you're familiar with a Layer 3 Clos, let's take it to the next level. The key to building a large Clos is the number of ports that are available in the spine. An easy way to increase the number of ports in the spine is to use VCF, as illustrated in Figure 1-34.

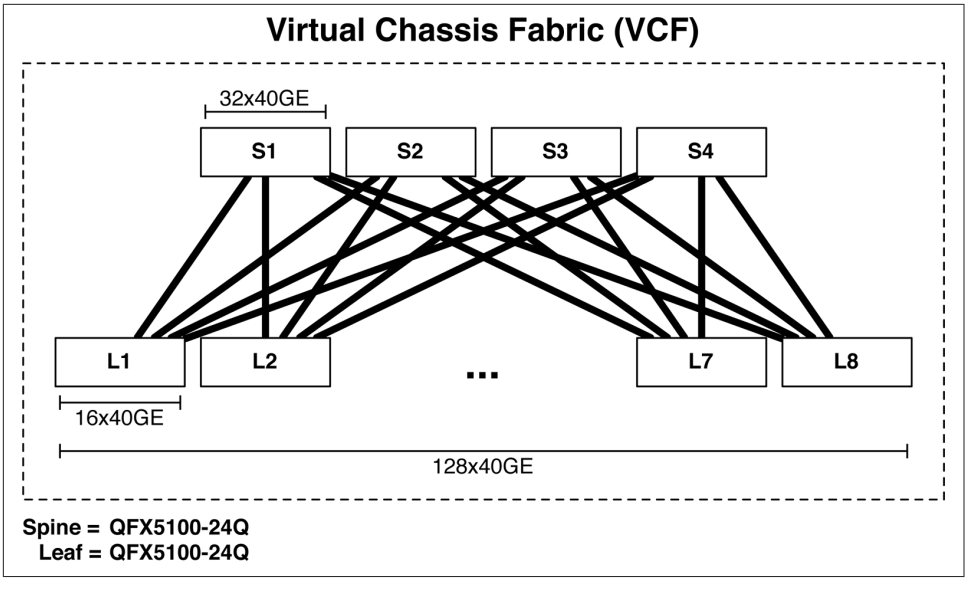


Figure 1-34. 128 40GbE VCF with 1:1 over-subscription

Using the 128 40GbE VCF illustrated in Figure 1-34, we've increased the capacity of a single spine switch from 32 to 128. Now, let's use the 128 40GbE VCF as a spine switch in a Clos fabric, as shown in Figure 1-35.

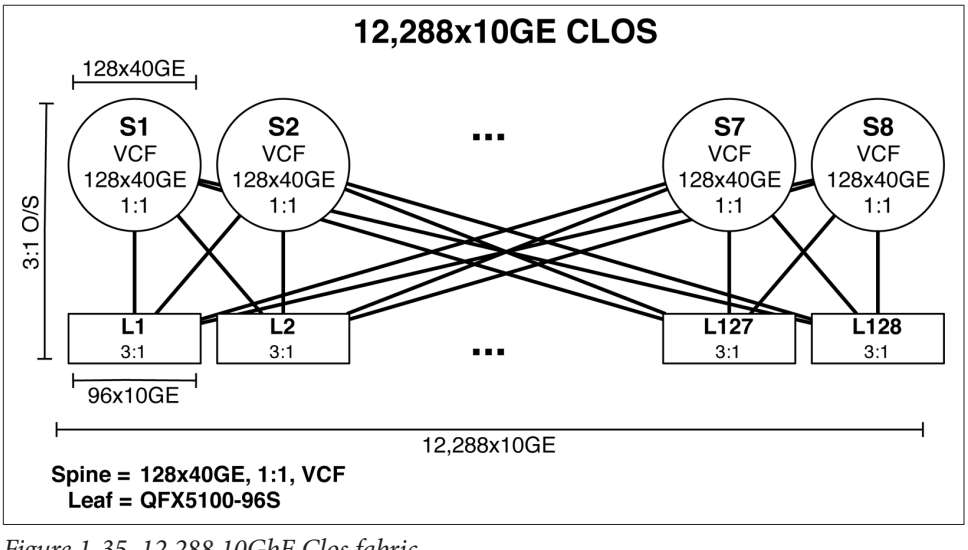


Figure 1-35. 12,288 10GbE Clos fabric

Something very interesting just happened in this design option: we've used a "feature within a feature" or a hierarchical approach. Typically, most people confine their

thinking to a single physical switch when identifying a spine switch. If you take a look at the requirements of a spine switch, it's fairly basic:

- Generally 1:1 over-subscription
- Generally 40GbE interfaces
- Single point of management
- Capable of Layer 3

It's true that a single physical switch meets the above requirements, but so does VCF. In the example in Figure 1-35 the spine switches are VCF fabrics. Each VCF spine has four QFX5100-24Q switches in its spine and eight QFX5100-24Q switches in its leaves; this gives a total of 128 40GbE interfaces at 1:1 over-subscription. To meet the 1:1 over-subscription requirement, each leaf has 16 40GbE down and 16 40GbE going up to the spine. Four spine switches handling 16×40GbE interfaces from each leaf can result in 8 leaves total (4 spines × 32 ports)/16 uplinks = 8 leaves. We can summarize the spines S1 through S8 in Figure 1-35 as follows:

- (4) QFX5100-24Q switches in the spine.
- (8) QFX5100-24-Q switches as the leaves.
  - 16 40GbE interfaces going up to the spine.
  - 16 40GbE interfaces available for use.
- 12 switches total in the VCF.
- 128 40GbE usable interfaces.
- 1:1 over-subscription.

The 12,288 10GbE design option has 138 discrete points of management; the assumption is that each VCF is a point of management (8) as well as of all the leaves (128). Here's a summary of the benefits of the Clos fabric:

- 12,288 10GbE ports
- No traffic loss during software upgrades with topology-independent ISSU
- Spine switches are a single point of management through VCF
- 3:1 over-subscription
- 2.5 µs end-to-end latency
- Lossless Ethernet support with PFC over Layer 3

Building a Clos is a great way to easily scale the network in an overlay architecture or in environments that don't require Layer 2 between leaves.

49,152 10GbE Clos

So, let's step up the game a little bit. Let's assume that we don't need a single point of management in the spine. How would this effect the scale of the network? Let's first start by building the largest 40GbE Clos spine we can with the Juniper QFX5100-24Q, assuming 1:1 over-subscription, as shown in Figure 1-36.

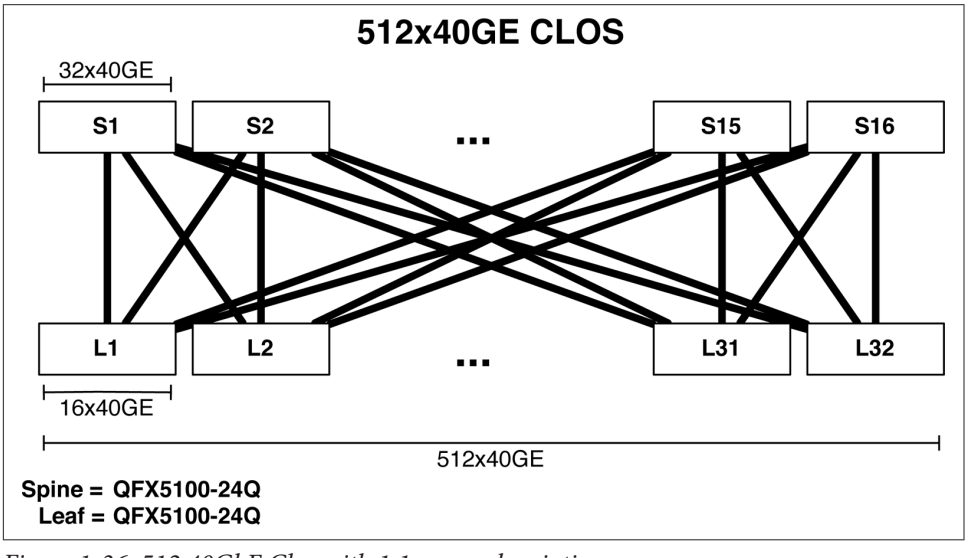


Figure 1-36. 512 40GbE Clos with 1:1 over-subscription

Now, each spine node can be a 512 40GbE Clos. This is the same concept as using VCF, but in this case, each switch in the Clos has to be managed separately; this type of design is referred as a *Clos within a Clos*.

<*meme*>Yo dawg, I herd you like Clos, so I put a Clos in your Clos so you can Clos while you Clos</*meme*>

With each spine supporting 512 40GbE interfaces, we can combine it with the Juniper QFX5100-96S switch, which has 8 40GbE uplinks. 512 ports × 8 spines / 8 uplinks = 512 leaves; finally 512 leaves × 96 ports = 49,152 10GbE ports, as illustrated in Figure 1-37.

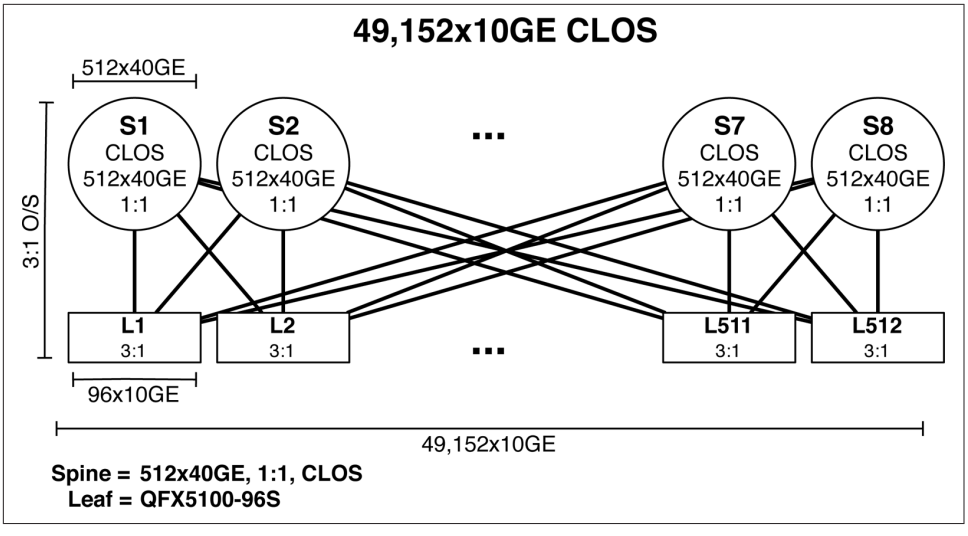


Figure 1-37. 49,152 10GbE Clos fabric

The end result is that we have a very large Clos fabric that supports 49,152 10GbE ports. Let's take a moment to review the benefits of the Clos fabric:

- 49,152 10GbE ports
- No traffic loss during software upgrades with topology-independent ISSU
- 3:1 over-subscription
- 2.5 µs end-to-end latency
- Lossless Ethernet support with PFC over Layer 3

One of the main assumptions to a Clos fabric of this size is that it operates at Layer 3; thus, your infrastructure and servers must not be dependent on Layer 2. Examples of such applications and infrastructure are web applications, Platform as a Service (PaaS), and overlay architecture. Keeping things simple at a large scale will reduce the amount of things that can go wrong. Layer 3 has very fast convergence, is loop free, and is able to scale easily.

Summary

This chapter has covered a lot of topics ranging from software to hardware. It's important to understand how the software and hardware are designed to work in conjunction with each other. This combination creates a best-in-class switch that is able to solve the difficult challenges data center operators are facing with the explosion of high-density 10GbE server ports and the need for delivering network services within seconds.

Being specifically designed to solve cloud computing and SDN requirements, the Juniper QFX5100 family solves a wide variety of challenges and offers many unique benefits, including the following:

Transport
: Dense 10GbE and 40GbE interfaces to build a deterministic spine-and-leaf topology with an option of 1:1, 3:1, or 6:1 over-subscription.

Interfaces
: Each 10GbE interface is tri-speed and supports 100 Mbps, 1GbE, or 10GbE. In addition, each interface can support either copper or fiber connectivity. Higher interface speeds such as 40GbE can be broken out into 4 10GbE interfaces by using a breakout cable.

Overlay Networking
: Each switch offers complete integration with Contrail and NSX to support overlay networking. You can configure the Juniper QFX5100 family as the end point in an overlay network architecture to support bare-metal servers.

Latency
: An intelligent algorithm is used for each ingress packet to determine which forwarding architecture—store-and-forward or cut-through—should be used to guarantee the least latency. On average, the port-to-port latency is only 500 nanoseconds.

Flexible Deployment Options
: The Juniper QFX5100 series doesn't force you into deploying a particular technology or proprietary protocol. The Juniper QFX5100 family supports standalone, Virtual Chassis, QFabric node, VCF, MC-LAG, or a Clos architecture.

QFabric Node
: You can use the Juniper QFX5100 as a node in the QFabric architecture. All of the benefits of the Juniper QFX5100 are available when used as a QFabric node: higher port density, overlay networking, and lower latency.

Virtualized Control Plane
: The Juniper QFX5100 takes virtualization to heart. The control plane uses an Intel Sandy Bridge CPU. The host operating system is Linux running KVM and QEMU for virtualization. The network operating system is Junos and runs as a VM and is able to take advantage of all of the benefits of virtualization such as ISSU.

Unified Forwarding Table
: Whether you need to support more MAC addresses or IPv4 prefixes in a Clos architecture, the Juniper QFX5100 allows you to adjust the profile of the for-

warding table. There are five preconfigured profiles that range from L2 heavy to L3 heavy.

Network Analytics
: Some applications are sensitive to microbursts and latency. The Juniper QFX5100 makes it possible for you to get on-box reporting of queue depth, queue latency, and microburst detection to facilitate and speed up the troubleshooting process.

Lossless Ethernet
: When converging storage and data, it's critical that storage be handled in such a way that no traffic is dropped. The Juniper QFX5100 family supports DCBX, ETS, and PFC to enable transit FCoE or lossless Ethernet for IP storage.

VCF
: Ethernet fabrics provide the benefit of a single point of management, lossless Ethernet, and full Layer 2 and Layer 3 services. The Juniper QFX5100 series can form a VCF Ethernet fabric. This is a spine-and-leaf topology that supports full ECMP but with all of the benefits of an Ethernet fabric.

Inline Network Services
: Traditionally network services such as GRE and NAT are handled by another device such as a router or firewall. The Juniper QFX5100 family can perform both GRE and NAT in hardware without a performance loss.

The Juniper QFX5100 family brings a lot new features and differentiation to the table when it comes to solving data center challenges. Because of the wide variety of features and differentiation, the Juniper QFX5100 is able to be positioned into many different types of architectures, such as the following:

High-Frequency Trading
: Speed is king when it comes to trading stocks; with an average port-to-port latency of 500 nanoseconds, the Juniper QFX5100 family fits well in a high-frequency trading architecture.

Private Cloud
: Although the Juniper QFX5100 was specifically designed to solve the challenges of cloud computing and public clouds, you can use the same features to solve the needs of the private cloud. Enterprises, government agencies, and research institutes are building out their own private clouds, and the Juniper QFX5100 meets and exceeds all requirements.

Campus
: High port density and a single point of management make the Juniper QFX5100 series a perfect fit in a campus architecture, specifically in the core and aggregation roles.

Enterprise
: Offering the flexibility to be used in multiple deployment scenarios, the Juniper QFX5100 family gives the enterprise the freedom to use the technology that best fits its needs. You can use it as a standalone device, Virtual Chassis, QFabric Node, VCF, MC-LAG, or in a Clos architecture.

It's a very exciting time in the networking industry as SDN, cloud computing, and data center technologies are continuing to push the envelope and bring new innovations and solutions to the field. The Juniper QFX5100 series of switches is embracing all of the change that's happening in the networking industry and providing clear and distinctive solution differentiation. With its wide variety of features and differentiation, the Juniper QFX5100 family is able to quickly solve the challenges of cloud computing in the data center as well as other use cases such as high-frequency trading and high-performance computing.

Chapter Review Questions

1. Which version of Junos is supported for three years?

a. The first major release of the year
b. The last maintenance release of the year
c. The last major release of the year
d. The last service release of the year

2. Which is not a function of the control plane?

a. Processing SSH traffic destined to the router
b. Updating the RIB
c. Updating the FIB
d. Processing a firewall filter on interface `xe-0/0/0.0`

3. How many modules does the Juniper QFX5100-24Q have?

a. 1
b. 2
c. 3
d. 4

4. Which functional block processes congestion management?

a. Intelligent parser
b. Traffic management
c. Buffer management

d. Frame modification

5. What hypervisor does the Juniper QFX5100 use for the control plane?

a. Microsoft Hyper-V

b. VMware ESXi

c. Linux KVM

d. Linux Containers

6. On what chipset is the Juniper QFX5100 based?

a. Broadcom Trident

b. Broadcom Trident II

c. Marvell Lion

d. Juniper Trio

7. On what tier in the network does the Juniper QFX5100 support ISSU?

a. Core

b. Aggregation

c. Access

d. All

Chapter Review Answers

*1. **Answer: C.** The last major release of Junos of a given calendar year is known as the Extended End of Life (EEOL) release and is supported for three years.*

*2. **Answer: D.** The data/forwarding plane handles all packet processing such as firewall filters, policers, or counters on the interface* `xe-0/0/0.0`.

*3. **Answer: B.** The Juniper QFX5100-24Q has two modules.*

*4. **Answer: C.** The buffer management block is responsible for all Quality of Service features, which includes congestion management.*

*5. **Answer: C.** The Juniper QFX5100 family uses the Linux KVM hypervisor to virtualize the control plane. Each VM runs the network operating system Junos.*

*6. **Answer: B.** The Juniper QFX5100 family is based on the Broadcom Trident II chipset.*

*7. **Answer: D.** Trick question. The Juniper QFX5100 family supports ISSU across platforms that can be positioned anywhere in the network.*

CHAPTER 2
Control Plane Virtualization

The key factors driving the Juniper QFX5100 are the advent of virtualization and cloud computing; however, there are many facets to virtualization. One is decoupling the service from the physical hardware. When this is combined with orchestration and automation, the service is now said to be *agile*: it has the ability to be quickly provisioned, even within seconds. Another aspect is scale in the number of instances of the service. Because it becomes so easy to provision a service, the total number of instances quickly increases.

Compute virtualization is such a simple concept, yet it yields massive benefit to both the end user and operator. The next logical step is to apply the benefits of compute virtualization to the control plane of the network. After all, the control board is nothing but an x86 processor, memory, and storage.

The immediate benefit of virtualizing the control board might not be so obvious. Generally, operators like to toy around and create a virtual machine (VM) running Linux so that they're able to execute operational scripts and troubleshoot. However, there is a much more exciting use case to virtualization of the control board. Traditionally, only networking equipment that was chassis-based was able to support two routing engines. The benefit of two routing engines is that it increases the high availability of the chassis and allows the operator to upgrade the control plane software in real time without traffic loss. This feature is commonly referred to as In-Service Software Upgrade (ISSU). One of the key requirements of ISSU is to have two routing engines that are synchronized using the Nonstop Routing (NSR), Nonstop Bridging (NSB), and Graceful Routing Engine Switchover (GRES) protocols. Fixed networking equipment such as top-of-rack (ToR) switches generally have only a single routing engine and do not support ISSU due to the lack of a second routing engine. Taking advantage of virtualization allows a ToR switch to have two virtualized routing engines that make possible features such as ISSU. The Juniper QFX5100 family takes

virtualization to heart and uses the Linux kernel-based virtual machine (KVM) as the host operating system and places Junos, the network operating system, inside of a VM. When an operator wants to perform a real-time software upgrade, the Juniper QFX5100 switch will provision a second routing engine, synchronize the data, and perform the ISSU without dropping traffic.

Another great benefit of compute virtualization inside of a switch is that you can create user-defined VMs and run your own applications and programs on the switch. Use cases include Network Functional Virtualization (NFV), network management, and statistical reporting.

Architecture

Recall that the Juniper QFX5100 series is split into two major components (see Figure 2-1): the control board and switch board. The control board is the foundation for the control plane, whereas the switch board is the foundation for the data plane.

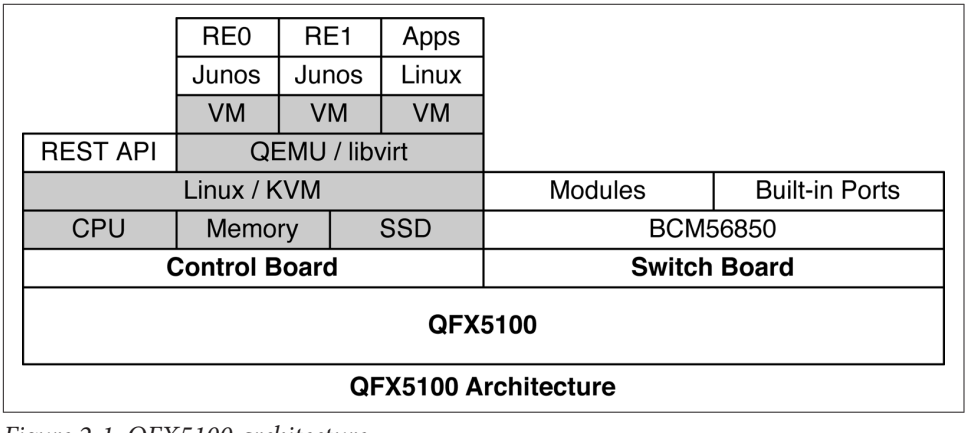


Figure 2-1. QFX5100 architecture

Focusing on the control board components, the blocks shaded in gray represent all of the roles in that architecture that are responsible for virtualizing the control plane. The control board is running commodity hardware that's easily compatible with common hypervisors. The processor is an Intel 1.5 Ghz dual-core Sandy Bridge CPU, and there is 8 GB of memory and a 32 GB solid-state disk (SSD). The Juniper QFX5100 boots directly into CentOS Linux instead of Junos; this provides the platform with several advantages. The first advantage is the ability to virtualize the underlying hardware by using Linux KVM and QEMU; the second advantage is the ability to host operational daemons and Application Programming Interfaces (APIs) directly on the host operating system.

To make the management of the hypervisor easier, the virtualization library (libvir) is used to provision and manage the VMs. The libvir provides a normalized management framework across a set of hypervisors. The ability to use a common framework to control a hypervisor provides more flexibility in the future if any of the underlying components happen to change.

Host Operating System

As mentioned in the previous section, the Juniper QFX5100 boots directly into Linux, specifically CentOS. This provides the operating system and virtualization foundation for Junos and all other network-related functionality.

Let's log in to the host operating system and do some exploring:

```
dhanks@qfx5100> request app-engine host-shell
Last login: Sun Nov 17 14:30:47 from 192.168.1.2
--- Host 13.2I20131114_1603_vsdk_build_30 built 2013-11-14 16:03:50 UTC
```

Now, let's take a peek at the PCI bus and see what's installed on the host operating system:

```
-sh-4.1# lspci
00:00.0 Host bridge: Intel Corporation 2nd Generation Core Processor Family DRAM Controller
(rev 09)
00:01.0 PCI bridge: Intel Corporation Xeon E3-1200/2nd Generation Core Processor Family PCI
Express Root Port (rev 09)
00:01.1 PCI bridge: Intel Corporation Xeon E3-1200/2nd Generation Core Processor Family PCI
Express Root Port (rev 09)
00:01.2 PCI bridge: Intel Corporation Xeon E3-1200/2nd Generation Core Processor Family PCI
Express Root Port (rev 09)
00:06.0 PCI bridge: Intel Corporation Xeon E3-1200/2nd Generation Core Processor Family PCI
Express Root Port (rev 09)
00:1c.0 PCI bridge: Intel Corporation DH89xxCC PCI Express Root Port #1 (rev 08)
00:1c.1 PCI bridge: Intel Corporation DH89xxCC PCI Express Root Port #2 (rev 08)
00:1c.2 PCI bridge: Intel Corporation DH89xxCC PCI Express Root Port #3 (rev 08)
00:1c.3 PCI bridge: Intel Corporation DH89xxCC PCI Express Root Port #4 (rev 08)
00:1d.0 USB controller: Intel Corporation DH89xxCC USB2 Enhanced Host Controller #1 (rev 08)
00:1f.0 ISA bridge: Intel Corporation DH89xxCC LPC Controller (rev 08)
00:1f.2 SATA controller: Intel Corporation DH89xxCC 4 Port SATA AHCI Controller (rev 08)
00:1f.3 SMBus: Intel Corporation DH89xxCC SMBus Controller (rev 08)
00:1f.7 System peripheral: Intel Corporation DH89xxCC Watchdog Timer (rev 08)
01:00.0 Co-processor: Intel Corporation Device 0434 (rev 21)
01:00.1 Ethernet controller: Intel Corporation DH8900CC Series Gigabit Network (rev 21)
01:00.2 Ethernet controller: Intel Corporation DH8900CC Series Gigabit Network (rev 21)
01:00.3 Ethernet controller: Intel Corporation DH8900CC Series Gigabit Network (rev 21)
01:00.4 Ethernet controller: Intel Corporation DH8900CC Series Gigabit Network (rev 21)
07:00.0 Unassigned class [ff00]: Juniper Networks Device 0062 (rev 01)
08:00.0 Unassigned class [ff00]: Juniper Networks Device 0063 (rev 01)
09:00.0 Ethernet controller: Broadcom Corporation Device b854 (rev 02)
```

Pretty vanilla so far. Four CPUs, a USB port, a SATA controller, and some network interface controllers (NICs). But, the two Juniper Networks devices are interesting; what are they? These are the FPGA controllers that are responsible for the chassis fan, sensors, and other environmental functions.

The final device is the Broadcom 56850 chipset. The way a network operating system controls the Packet Forwarding Engine (PFE) is simply through a PCI interface by using a Software Development Kit (SDK).

Let's take a closer look at the CPU:

```
-sh-4.1# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Pentium(R) CPU  @ 1.50GHz
stepping        : 7
cpu MHz         : 1500.069
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni
pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1
sse4_2 x2apic popcnt aes xsave avx lahf_lm arat epb xsaveopt pln pts dts tpr_shadow
vnmi flexpriority ept vpid
bogomips        : 3000.13
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Pentium(R) CPU  @ 1.50GHz
stepping        : 7
cpu MHz         : 1500.069
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 2
apicid          : 1
initial apicid  : 1
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni
```

```
pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1
sse4_2 x2apic popcnt aes xsave avx lahf_lm arat epb xsaveopt pln pts dts tpr_shadow
vnmi flexpriority ept vpid
bogomips        : 3000.13
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 2
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Pentium(R) CPU  @ 1.50GHz
stepping        : 7
cpu MHz         : 1500.069
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni
pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1
sse4_2 x2apic popcnt aes xsave avx lahf_lm arat epb xsaveopt pln pts dts tpr_shadow
vnmi flexpriority ept vpid
bogomips        : 3000.13
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 3
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Pentium(R) CPU  @ 1.50GHz
stepping        : 7
cpu MHz         : 1500.069
cache size      : 3072 KB
physical id     : 0
siblings        : 4
core id         : 1
cpu cores       : 2
apicid          : 3
initial apicid  : 3
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
```

```
constant_tsc arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni
pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm sse4_1
sse4_2 x2apic popcnt aes xsave avx lahf_lm arat epb xsaveopt pln pts dts tpr_shadow
vnmi flexpriority ept vpid
bogomips        : 3000.13
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
```

The CPU is a server-class Intel Xeon E3-1200 processor; it's a single socket with four cores. There's plenty of power to operate multiple VMs and the network operating system.

Now, let's move on to the memory:

```
-sh-4.1# free
             total       used       free     shared    buffers     cached
Mem:       7529184    3135536    4393648          0     158820     746800
-/+ buffers/cache:    2229916    5299268
Swap:
```

After some of the memory has been reserved by other hardware and the kernel, you can see that we have about 7.3 GB total.

Next, let's see how many disks there are and how they're partitioned:

```
-sh-4.1# fdisk -l

Disk /dev/sdb: 16.0 GB, 16013852672 bytes
255 heads, 63 sectors/track, 1946 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000dea11

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1         125     1000000   83  Linux
Partition 1 does not end on cylinder boundary.
/dev/sdb2             125        1857    13914062+  83  Linux

Disk /dev/sda: 16.0 GB, 16013852672 bytes
255 heads, 63 sectors/track, 1946 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000d8b25

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1         125     1000000   83  Linux
Partition 1 does not end on cylinder boundary.
/dev/sda2             125        1857    13914062+  83  Linux

Disk /dev/mapper/vg0_vjunos-lv_junos_recovery: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

```
Disk /dev/mapper/vg0_vjunos-lv_var: 11.3 GB, 11307843584 bytes
255 heads, 63 sectors/track, 1374 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/mapper/vg0_vjunos-lv_junos: 12.9 GB, 12884901888 bytes
255 heads, 63 sectors/track, 1566 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

The host system has two SSD storage devices, each with 16 GB of capacity. From the partition layout illustrated in Figure 2-2, you can see that we're running the Linux Volume Manager (LVM).

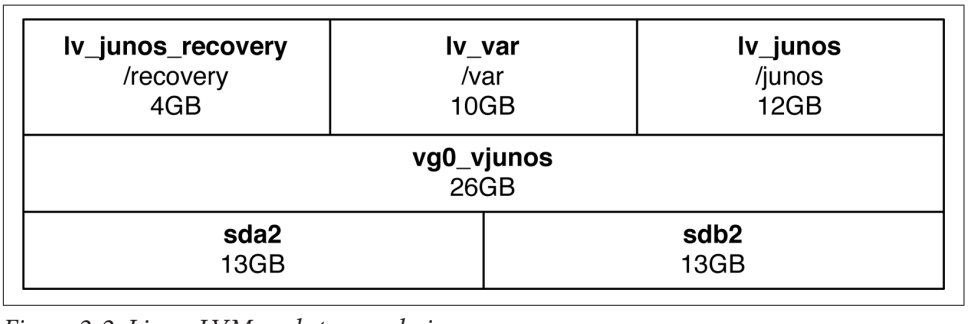


Figure 2-2. Linux LVM and storage design

There are two 16 GB SSDs, which are part of the Linux LVM. The primary volume group is vg0_vjunos. This volume group has three volumes that are used by Junos:

- lv_junos_recovery
- lv_var
- lv_junos

Linux KVM

When the Juniper QFX5100 boots up, the host operating system is Linux. All of the control plane operations happen within the network operating system, Junos. The Juniper QFX5100 takes advantage of compute virtualization in the host operating system by using Linux KVM. A VM is created specifically for Junos. Given that KVM can create multiple VMs, the Juniper QFX5100 series has the ability to perform ISSU and support third-party VMs that can host additional services such as network management and monitoring.

virsh

The Juniper QFX5100 uses the libvir library as well as the `libsh` management user interface to interact with Linux KVM. If you're familiar with libvir, walking around the virtualization capabilities of the Juniper QFX5100 will come as second nature. If you aren't familiar with libvir, let's use `virsh` to explore and see what's happening under the hood.

The first thing we need to do is drop into the host shell from the Junos CLI:

```
dhanks@qfx5100> request app-engine host-shell
Last login: Sun Nov 17 14:30:47 from 192.168.1.2
--- Host 13.2I20131114_1603_vsdk_build_30 built 2013-11-14 16:03:50 UTC
```

Now, let's take a look at the VMs installed in the Linux KVM:

```
-sh-4.1# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     vjunos0                        running
```

By default there's a single VM running the Junos networking operating system. The VM's name is `vjunos0` with an ID of 1, and we can see that the state is `running`.

Hmm. Are you curious as to what version of the libvir library and QEMU the Juniper QFX5100 is using? Let's find out:

```
-sh-4.1# virsh version
Compiled against library: libvir 0.9.10
Using library: libvir 0.9.10
Using API: QEMU 0.9.10
Running hypervisor: QEMU 0.12.1
```

At this point, let's take a look at the overall host memory and CPU statistics:

```
-sh-4.1# virsh nodememstats
total  :              7269088 kB
free   :              4147596 kB
buffers:               264772 kB
cached :               761476 kB

-sh-4.1#
-sh-4.1# virsh nodecpustats

user:                305995340000000
system:              145678380000000
idle:              11460475070000000
iowait:               1075190000000
sdf
```

Now that we're familiar with what the host system is capable of, software versions, and of course how many VMs are configured, let's examine the Junos VM:

```
-sh-4.1# virsh dominfo vjunos0
Id:             1
Name:           vjunos0
UUID:           100e7ead-ae00-0140-0000-564a554e4f53
```

```
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       445895.2s
Max memory:     2000896 kB
Used memory:    2000896 kB
Persistent:     no
Autostart:      disable
Managed save:   no
```

Each VM has a unique identifier that can be used to refer to the VM. One of the more interesting attributes is the OS Type, which is set to `hvm`; this stands for Hardware Virtual Machine. Because Junos is based on FreeBSD and heavily modified to support network control plane functions, it's difficult to say that it's pure FreeBSD. Instead, the alternative is to use a vendor-neutral OS Type of `hvm`, which basically means that it's an x86-based operating system.

Let's focus on the memory and network settings for `vjunos0`:

```
-sh-4.1# virsh dommemstat vjunos0
rss 1895128

-sh-4.1# virsh domiflist vjunos0
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      bridge     virbr0     e1000       52:54:00:bf:d1:6c
vnet1      bridge     ctrlbr0    e1000       52:54:00:e7:b6:cd
```

In the 13.2X53D20 version of Junos, there are two bridges installed for the VMs within KVM. The `vnet0`/`virbr0` interface is used across all of the VMs to communicate with the outside world through their management interfaces. The other interface, `vnet1`/`ctrlbr0`, is used exclusively for ISSU. During an ISSU, there are two copies of Junos running; all control plane communication between the VMs are performed over this special bridge so that any other control plane functions such as Secure Shell (SSH), Open Shortest Path First (OSPF), and Border Gateway Protocol (BGP) aren't impacted while synchronizing the kernel state between the master and backup Junos VMs.

Another interesting place to look for more information is in the */proc* filesystem. We can take a look at the process ID (PID) of `vjunos0` and examine the task status:

```
-sh-4.1# cat /var/run/libvirt/qemu/vjunos0.pid
2972
-sh-4.1# cat /proc/2972/task/*/status
Name:   qemu-kvm
State:  S (sleeping)
Tgid:   2972
Pid:    2972
PPid:   1
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
Utrace: 0
FDSize: 256
```

```
Groups:
VmPeak:  2475100 kB
VmSize:  2276920 kB
VmLck:         0 kB
VmHWM:   1895132 kB
VmRSS:   1895128 kB
VmData:  2139812 kB
VmStk:        88 kB
VmExe:      2532 kB
VmLib:     16144 kB
VmPTE:      4284 kB
VmSwap:        0 kB
Threads:        2
SigQ:   1/55666
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000010002840
SigIgn: 0000000000001000
SigCgt: 0000002180006043
CapInh: 0000000000000000
CapPrm: ffffffffc00000000
CapEff: ffffffffc00000000
CapBnd: ffffffffc00000000
Cpus_allowed:   04
Cpus_allowed_list:      2
Mems_allowed:
00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        5825006750
nonvoluntary_ctxt_switches:     46300
Name:   qemu-kvm
State:  S (sleeping)
Tgid:   2972
Pid:    2975
PPid:   1
TracerPid:      0
Uid:    0       0       0       0
Gid:    0       0       0       0
Utrace: 0
FDSize: 256
Groups:
VmPeak:  2475100 kB
VmSize:  2276920 kB
VmLck:         0 kB
VmHWM:   1895132 kB
VmRSS:   1895128 kB
VmData:  2139812 kB
VmStk:        88 kB
VmExe:      2532 kB
VmLib:     16144 kB
VmPTE:      4284 kB
VmSwap:        0 kB
Threads:        2
SigQ:   1/55666
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: fffffffde7ffbfebf
```

```
SigIgn: 0000000000001000
SigCgt: 0000002180006043
CapInh: 0000000000000000
CapPrm: ffffffffc00000000
CapEff: ffffffffc00000000
CapBnd: ffffffffc00000000
Cpus_allowed:   04
Cpus_allowed_list:      2
Mems_allowed:
00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000000,00000000,00000000,00000000,00000000,00000000,
00000000,00000001
Mems_allowed_list:      0
voluntary_ctxt_switches:        5526311517
nonvoluntary_ctxt_switches:     586609665
```

One of the more interesting things to notice is the `Cpus_allowed_list`, which is set to a value of 2. By default, Juniper assigns the third CPU directly to the `vjunos0` VM; this guarantees that other tasks outside of the scope of the control plane don't negatively impact Junos. The value is set to 2 because the first CPU has a value of 0. We can verify this again with another `virsh` command:

```
-sh-4.1# virsh vcpuinfo vjunos0
VCPU:           0
CPU:            2
State:          running
CPU time:       311544.1s
CPU Affinity:   --y-
```

We can see that the CPU affinity is set to `y` on the third CPU, which verifies what we see in the */proc* file system.

App Engine

If you're interested in learning more about the VMs but don't feel like dropping to the host shell and using `virsh` commands, there is an alternative called the Junos App Engine, which is accessible within the Junos CLI.

To view the App Engine settings, use the `show app-engine` command. There are several different views that are available, as listed in Table 2-1.

Table 2-1. Junos App Engine views

View	Description
ARP	View all of the ARP entries of the VMs connected into all the bridge domains
Bridge	View all of the configured Linux bridge tables
Information	Get information about the compute cluster, such as model, kernel version, and management IP addresses
Netstat	Just a simple wrapper around the Linux `netstat -rn` command
Resource usage	Show the CPU, memory, disk, and storage usage statistics in an easy-to-read format

Let's explore some of the most common Junos App Engine commands and examine the output:

```
dhanks@QFX5100> show app-engine arp
Compute cluster: default-cluster

  Compute node: default-node

   Arp
   ===
   Address                 HWtype  HWaddress        Flags Mask            Iface
   192.168.1.2             ether   10:0e:7e:ad:af:30   C                 virbr0
```

This is just a simple summary `show` command that aggregates the management IP, MAC, and the bridge table to which it's bound.

Let's take a look at the bridge tables:

```
dhanks@QFX5100> show app-engine bridge
Compute cluster: default-cluster

  Compute node: default-node

   Bridge Table
   ============
   bridge name  bridge id               STP enabled     interfaces
   ctrlbr0              8000.fe5400e7b6cd       no      vnet1
   virbr0               8000.100e7eadae03       yes     virbr0-nic
                                                        vnet0
```

Just another nice wrapper for the Linux `brctl` command. Recall that `vnet0` is for the regular control plane side of Junos, whereas `vnet1` is reserved for inter-routing engine traffic during an ISSU:

```
dhanks@QFX5100> show app-engine resource-usage
Compute cluster: default-cluster

  Compute node: default-node
   CPU Usage
   =========
   15:48:46     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest   %idle
   15:48:46     all    0.30    0.00    1.22    0.01    0.00    0.00    0.00    2.27   96.20
   15:48:46       0    0.08    0.00    0.08    0.03    0.00    0.00    0.00    0.00   99.81
   15:48:46       1    0.08    0.00    0.11    0.00    0.00    0.00    0.00    0.00   99.81
   15:48:46       2    1.03    0.00    4.75    0.01    0.00    0.00    0.00    9.18   85.03
   15:48:46       3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00

   Memory Usage
   ============
                total       used       free     shared    buffers     cached
   Mem:          7098       3047       4051          0        258        743
   Swap:            0          0          0

   Disk Usage
   ==========
   Filesystem            Size  Used Avail Use% Mounted on
   tmpfs                 3.5G  4.0K  3.5G   1% /dev/shm
   /dev/mapper/vg0_vjunos-lv_var
```

```
                       11G  198M  9.7G   2% /var
/dev/mapper/vg0_vjun
os-lv_junos
                       12G  2.2G  9.1G  20% /junos
/dev/mapper/vg0_vjunos-lv_junos_recovery
                      4.0G  976M  2.8G  26% /recovery
/dev/sda1             962M  312M  602M  35% /boot

Storage Information
===================
  VG         #PV #LV #SN Attr   VSize  VFree
  vg0_vjunos   2   3   0 wz--n- 26.53g    0
```

`show app-engine resource-usage` is a nice aggregated command showing the utilization of the CPU, memory, disk, and storage information; it's a very easy way to get a bird's-eye view of the health of the App Engine.

ISSU

Since the original M Series routers, one of the great Junos features is its ability to support ISSU. With ISSU, the network operating system can upgrade the firmware of the router without having to shut it down and impact production traffic. One of the key requirements for ISSU is that there are two routing engines. During an ISSU, the two engines need to synchronize kernel and control plane state with each other. The idea is that one routing engine is upgraded while the other routing engine is handling the control plane.

Although Juniper QFX5100 switches don't physically have two routing engines, they are able to carry out the same functional requirements thanks to the power of virtualization. The Juniper QFX5100 series is able to create a second VM running Junos during an ISSU to meet all of the synchronization requirements, as is illustrated in Figure 2-3.

Each Junos VM has three management interfaces. Two of those interfaces, `em0` and `em1`, are used for management and map to the external interfaces `C0` and `C1`, respectively. The third management interface, `em2`, is used exclusively for communication between the two Junos VMs. For example, control plane protocols such as NSR, NSB, and GRES are required in order for a successful ISSU to complete; these protocols would communicate across the isolated `em2` interface as well as an isolated `ctrlbr0` bridge table in the Linux host.

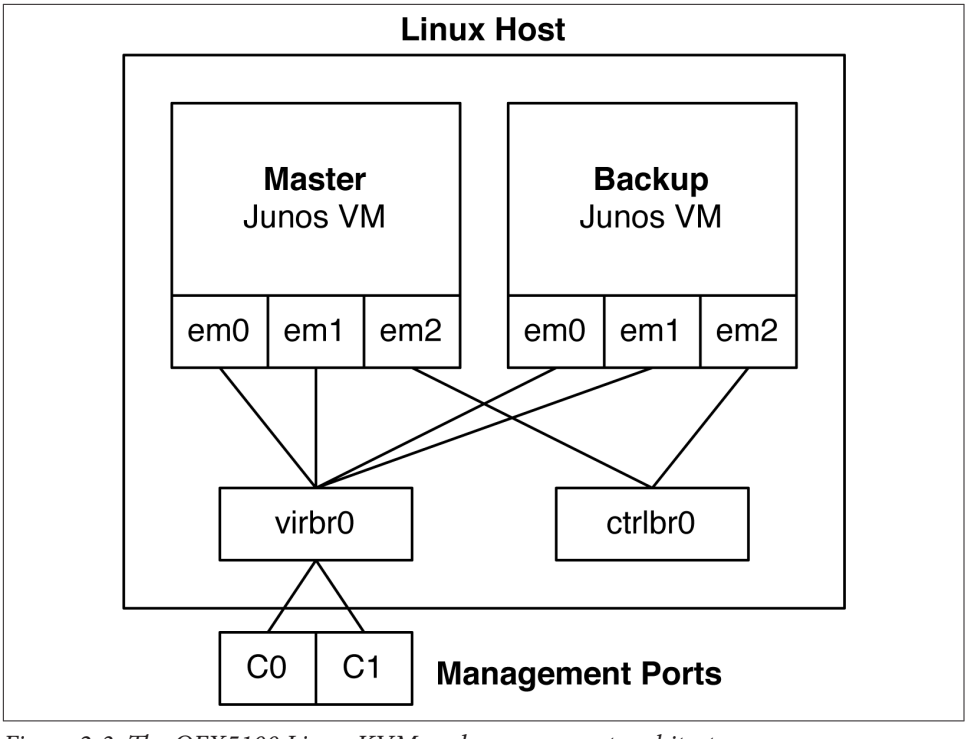


Figure 2-3. The QFX5100 Linux KVM and management architecture

The backup Junos VM is only created and running during an ISSU. At a high level, Junos goes through the following steps during an ISSU:

- The backup Junos VM is created and started.
- The backup Junos VM is upgraded to the software version specified in the ISSU command.
- The PFE goes into an ISSU-prepared state in which data is copied from the PFE to RAM.
- The PFE connects to the recently upgraded backup Junos VM, which now becomes the master routing engine.
- The PFE performs a warm reboot.
- The new master Junos VM installs the PFE state from RAM back into the PFE.
- The other Junos VM is shut down.
- Junos has been upgraded and the PFE has performed a warm reboot.

Let's see an ISSU in action:

```
dhanks@QFX5100> request system software in-service-upgrade flex-13.2X51-D20.2-
domestic-signed.tgz
warning: Do NOT use /user during ISSU. Changes to /user during ISSU may get lost!
ISSU: Validating Image
error: 'Non Stop Routing' not configured
error: aborting ISSU
error: ISSU Aborted!
ISSU: IDLE
```

Ah, bummer! What happened here? There are some requirements for the control plane that must be enabled before a successful ISSU can be achieved:

- NSR
- NSB
- GRES
- Commit Synchronization

Let's configure these quickly and try an ISSU once again.

```
{master:0}[edit]
dhanks@QFX5100# set chassis redundancy graceful-switchover
{master:0}[edit]
dhanks@QFX5100# set protocols layer2-control nonstop-bridging
{master:0}[edit]
dhanks@QFX5100# set system commit synchronize
{master:0}[edit]
dhanks@QFX5100# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode
```

OK, now that all of the software features required for ISSU are configured and committed, let's try the ISSU one more time:

```
dhanks@QFX5100> request system software in-service-upgrade flex-13.2X51-D20.2-
domestic-signed.tgz
warning: Do NOT use /user during ISSU. Changes to /user during ISSU may get lost!
ISSU: Validating Image
ISSU: Preparing Backup RE
Prepare for ISSU
ISSU: Backup RE Prepare Done
Extracting jinstall-qfx-5-flex-13.2X51-D20.2-domestic ...
Install jinstall-qfx-5-flex-13.2X51-D20.2-domestic completed
Spawning the backup RE
Spawn backup RE, index 1 successful
GRES in progress
GRES done in 0 seconds
Waiting for backup RE switchover ready
GRES operational
Copying home directories
Copying home directories successful
Initiating Chassis In-Service-Upgrade
Chassis ISSU Started
ISSU: Preparing Daemons
ISSU: Daemons Ready for ISSU
ISSU: Starting Upgrade for FRUs
```

```
ISSU: Preparing for Switchover
ISSU: Ready for Switchover
Checking In-Service-Upgrade status
  Item           Status                  Reason
  FPC 0          Online
Send ISSU done to chassisd on backup RE
Chassis ISSU Completed
ISSU: IDLE
Initiate em0 device handoff
pci-stub 0000:01:00.1: transaction is not cleared; proceeding with reset anyway
pci-stub 0000:01:00.1: transaction is not cleared; proceeding with reset anyway
pci-stub 0000:01:00.1: transaction is not cleared; proceeding with reset anyway
pci-stub 0000:01:00.1: transaction is not cleared; proceeding with reset anyway
em0: bus=0, device=3, func=0, Ethernet address 10:0e:7e:b2:2d:78
hub 1-1:1.0: over-current change on port 1
hub 1-1:1.0: over-current change on port 3
hub 1-1:1.0: over-current change on port 5

QFX5100 (ttyd0)

login:
```

Excellent! The ISSU has completed successfully and no traffic was impacted during the software upgrade of Junos.

One of the advantages of the Broadcom warm reboot feature is that no firmware is installed in the PFE. This effectively makes the ISSU problem a control plane–only problem, which is very easy to solve. When you need to synchronize both the PFE firmware and control plane firmware, there are more moving parts, and the problem is more difficult to solve. *Juniper MX Series* by Douglas Richard Hanks, Jr. and Harry Reynolds (O'Reilly) thoroughly explains all of the benefits and drawbacks of ISSU in such a platform that upgrades both the control plane firmware in addition to the PFE firmware. The end result is that a control plane–only ISSU is more stable and finishes much faster when compared to a platform such as the Juniper MX. However, the obvious drawback is that no new PFE features can be used as part of a control plane–only ISSU, which is where the Juniper MX would win.

Summary

This chapter walked you through the design of the control plane and how the Juniper QFX5100 is really just a server that thinks it's a switch. The Juniper QFX5100 has a powerful Intel CPU, standard memory, and SSD hard drives. What was surprising is that the switch boots directly into Linux and uses KVM to virtualize Junos, which is the network operating system. Because Junos is running a VM, it enables the Juniper QFX5100 to support carrier-class features such as ISSU, NSR, and NSB.

CHAPTER 3

Performance and Scaling

One of the more challenging tasks of a network architect is to ensure that a design put forth meets the end-to-end solution requirements. The first step is identifying all of the roles in an architecture; this could be as simple as defining the edge, core, aggregation, and access tiers in the network. Each role has a specific set of responsibilities in terms of functionality and requirements. To map a product to a role in an architecture, the product must meet or exceed the requirements and functionality required by each role for which it's being considered. Thus, building an end-to-end solution architecture is a bit like a long chain: it's only as strong as the weakest link.

The most common method for ascertaining the product capabilities, performance, and scale are through datasheets or the vendor's account team. However, the best method is actually testing by going through a proof of concept or certification cycle. This requires that you build out all of the roles and products in the architecture and measure the end-to-end results; this method quickly flushes out any issues before moving into procurement and production.

This chapter will walk through all of the performance and scaling considerations required to successfully map a product into a specific role in an end-to-end architecture. Attributes such as MAC address, host entries, and IPv4 prefixes will be clearly spelled out. Armed with this data, you will be able to easily map Juniper QFX5100 series switches into many different roles in your existing network.

Design Considerations

Before any good network architect jumps head first into performance and scaling requirements, he will need to make a list of design considerations. Each one places an additional tax on the network that is outside of the scope of traditional performance and scaling requirements.

Overlay Architecture

One of the first design questions that you need to consider when planning a next-generation network is do you need to centrally orchestrate all resources in the data center so that applications can be deployed within seconds? The follow-up question is do you currently virtualize your data center compute and storage with hypervisors and cloud management platforms? If the answer is yes to both these questions, you must consider an *overlay architecture* when it comes to the data center network.

Given that compute and storage has already been virtualized, the next step is to virtualize the network. By using an overlay architecture in the network, you can decouple physical hardware from the network, which is one of the primary tenets of virtualization. Decoupling the network from the physical hardware allows the network to be programmatically provisioned within seconds. As of this writing, two great examples of products that support overlay architectures are Juniper Contrail and VMware NSX.

Moving to a new network architecture places a different "network tax" on the data center. Traditionally, when servers and virtual machines (VMs) are connected to a network, they each consume a MAC address and host route entry in the network. However, in an overlay architecture, only the virtual tunnel end points (VTEP) consume a MAC address and host route entry in the network. All VM traffic is now encapsulated between VTEPs and the MAC address, and the host route of each VM isn't visible to the underlying networking equipment. Now, the MAC address and host route scale has been moved from the physical network hardware to the hypervisor.

Bare-metal servers

It's rare to find a data center that has virtualized 100 percent of its compute resources. There's always a subset of servers that you cannot virtualize due to performance, compliance, or any number of other reasons. This raises an interesting question: if 80 percent of the servers in the data center are virtualized and take advantage of an overlay architecture, how do you provide connectivity to the other 20 percent of physical servers?

Overlay architectures support several mechanisms to provide connectivity to physical servers. The most common option is to embed a VTEP into the physical access switch, as demonstrated in Figure 3-1.

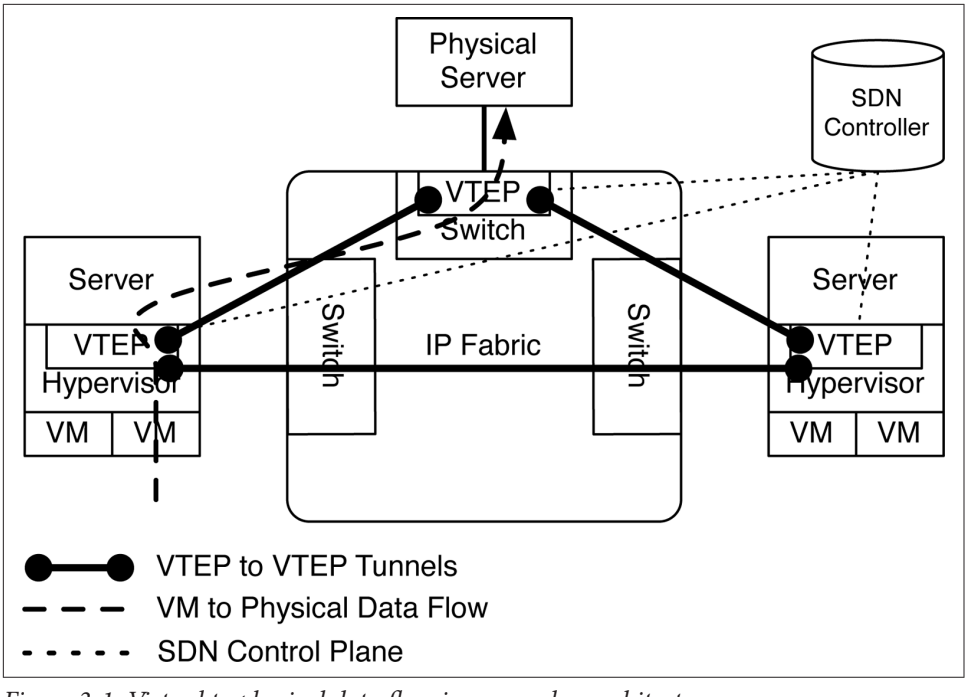


Figure 3-1. Virtual to physical data flow in an overlay architecture

In Figure 3-1, each server on the left and right of the IP Fabric has been virtualized with a hypervisor. Each hypervisor has a VTEP within it that handles the encapsulation of data plane traffic between VMs. Each VTEP also handles MAC address learning, provisioning of new virtual networks, and other configuration changes. The server on top of the IP Fabric is a simple physical server but doesn't have any VTEP capabilities of its own. For the physical server to participate in the overlay architecture, it needs something to encapsulate the data plane traffic and perform MAC address learning. Being able to handle the VTEP role inside of an access switch simplifies the overlay architecture. Now, each access switch that has physical servers connected to it can simply perform the overlay encapsulation and control plane on behalf of the physical server. From the point of view of the physical server, it simply sends traffic into the network without having to worry about anything else.

The Juniper QFX5100 series supports full overlay integration for both Juniper Contrail and VMware NSX in the data plane and control plane. However, the use case isn't limited to only bare-metal servers; another use case would be to inject physical network services such as load balancing or firewalls into an overlay architecture.

Juniper Architectures versus Open Architectures

The other common design option is to weight the benefits of Juniper architectures with open architectures. The benefits of a Juniper architecture is that it has been designed specifically to enable turnkey functionality, but the downside is that it requires a certain set of products to operate. On the other side are open architectures. The benefit to an open architecture is that it can be supported across a set of multiple vendors, but the downside is that you might lose some capabilities that are only available in the Juniper architectures.

Generally, it boils down to the size of the network. If you know that your network will never grow past a certain size and you're procuring all of the hardware up front, using a Juniper architecture might simply outweigh all of the benefits of an open architecture, because there isn't a need to support multiple vendors. Another scenario is that your network is large enough that you can't build it all at once and want a pay-as-you-grow option over the next five years. A logical option would be to implement open architectures so that as you build out your network, you aren't limited in the number of options going forward. Another option would be to take a hybrid approach and build out the network in points of delivery (POD). Each POD could have the option to take advantage of proprietary architectures or not.

Each business and network is going to have any number of external forces that weigh on the decision to go with Juniper architectures and open architectures, and more often than not, these decisions change over time. Unless you know 100 percent of these nuances up front, it's important to select a networking platform that offers both Juniper architectures and open architectures.

The Juniper QFX5100 series offers the best of both worlds. It supports open architectures equally as well as Juniper architectures, as is summarized here:

Juniper Architectures
: The Juniper QFX5100 family is able to participate in a Juniper QFabric architecture as a node. You can also use them to build a Virtual Chassis Fabric (VCF) or a traditional Virtual Chassis. In summary, these Juniper architectures give you the ability to build a plug-and-play Ethernet fabric with a single point of management and support converged storage.

Open Architectures
: Juniper QFX5100 switches support Multi-Chassis Link Aggregation (MC-LAG) so that downstream devices can simply use IEEE 802.1AX/LACP to connect and transport data. The Juniper QFX5100 series also supports a wide range of open protocols, such as Border Gateway Protocol (BGP), Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), and a suite of Multiprotocol Label Switching (MPLS) technologies.

The Juniper QFX5100 makes a great choice no matter where you place it in your network. You could choose to deploy an open architecture today, and change to a Juniper architecture in the future. One of the best tools in creating a winning strategy is to keep the number of options high.

Over-subscription

There are several different types of chipsets in the Broadcom Trident II family. Each chipset has different performance and over-subscription values. Table 3-1 lists them for you.

Table 3-1. Broadcom Trident II family bandwidth and over-subscription options

Broadcom chipset	I/O bandwidth	Core bandwidth	Over-subscription ratio
Trident II: option 1	1,280 Gbps	960 Gbps	4:3
Trident II: option 2	1,280 Gbps	720 Gbps	16:9
Trident II: option 3	960 Gbps	960 Gbps	1:1
Trident II: option 4	720 Gbps	720 Gbps	1:1

All of the Juniper QFX5100 platforms have been designed around Broadcom Trident II option 1, which is the BCM56850 chipset. Out of all of the options available, this chipset represents the most I/O and core bandwidth available. To fully understand the implications of the 4:3 over-subscription, let's take a closer look at the chipset's architecture.

Architecture

The BCM56850 is divided into four groups (see Figure 3-2). Each group supports 25% of the available core bandwidth, which in the case of the BCM56850 is 960 Gbps; thus, each group supports 240 Gbps in the core. Each group also has a set of eight cores that are responsible for processing traffic. Each core can handle 40 Gbps of traffic, and because each group has eight cores, the total amount of I/O bandwidth each group can support is 320 Gbps.

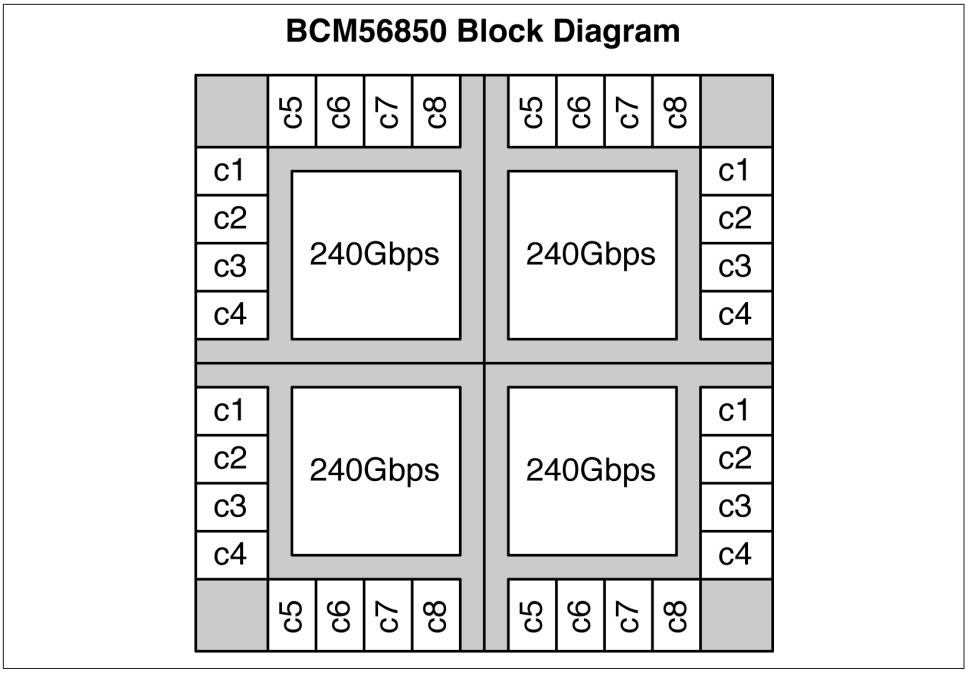


Figure 3-2. Block diagram of the BCM58850 chipset

In summary, each group supports 240 Gbps of core bandwidth and 320 Gbps of I/O bandwidth via the eight cores. Simplifying the ratio 320:240 results in the 4:3 oversubscription, as stipulated earlier in Table 3-1.

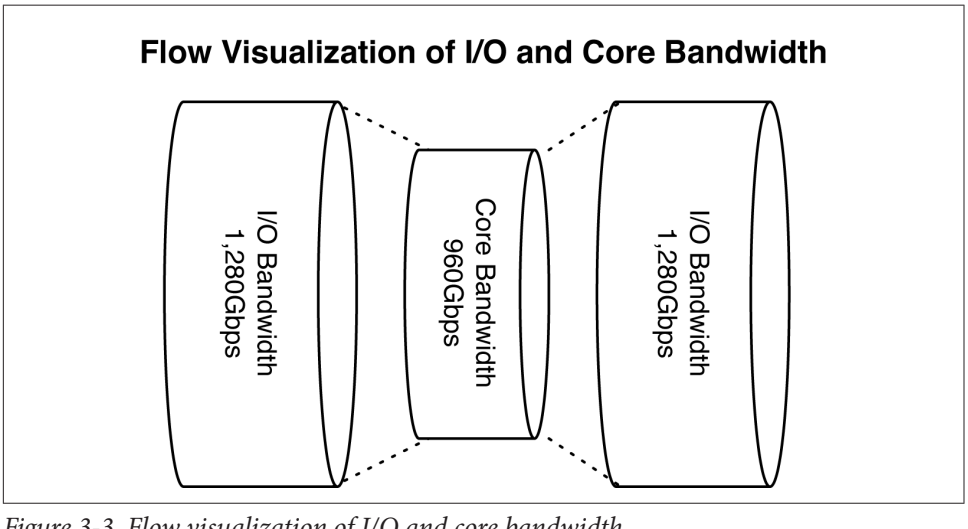


Figure 3-3. Flow visualization of I/O and core bandwidth

The final result in an over-subscription of the I/O to core bandwidth is that packets of a certain size will be dropped assuming that all of the ports in the switch are running at line rate. Details of the effects of over-subscription are discussed in the "Performance" on page 84 later in the chapter.

QFX5100-24Q System Modes

As a result of the over-subscription and port channelization features of the BCM56850 chipset, the data center operator is afforded more flexibility in the deployment of the switch. The Juniper QFX5100-24Q is the most flexible platform in the Juniper QFX5100 series, and it supports several system modes in which the switch can operate. Each mode is designed specifically to enable certain capabilities over the others. Understanding what each mode enables is critical because it will be another design consideration in the overall architecture of your network.

Any renumbering of interfaces requires a warm Broadcom chipset reboot. For example, changing from one mode to another will cause a small interruption in data plane traffic as the Broadcom chipset performs a warm reboot to reconfigure the number of ports. The only exception is the Flexible QIC mode. Depending on which QIC you use, the number of ports can vary; however, as long as you stay in Flexible QIC mode, no Broadcom chipset reboot is required.

Fully subscribed mode

The fully subscribed mode is the default mode for the Juniper QFX5100-24Q. Because the Juniper QFX5100-24Q has a native bandwidth capacity of 960 Gbps (24 ports of 40 Gbps) without any modules installed, it's able to provide full line-rate performance for all packet sizes without drops. In this default mode, you cannot use any of the QIC modules; however, you can channelize all of the native 40GbE ports into 4 10GbE interfaces. The port configurations can be summarized as follows:

24 40GbE
: In the default configuration, you can use all of the 40GbE interfaces on the Juniper QFX5100-24Q.

96 10GbE
: By taking advantage of port channelizing, each of the 40GbE interfaces can be broken out into 4 10GbE interfaces.

In summary, the default mode only supports the 24 40GbE interfaces on the Juniper QFX5100-24Q; you cannot use the two QIC modules.

104-port mode

One of the limitations of the BCM56850 chipset is that the total port count cannot exceed 104. For such a scenario in which you require 104 10GbE interfaces, the Juniper QFX5100-24Q can be put into a 104-port system mode. It's required that you channelize each of the native 24 40GbE interfaces. In addition, this mode requires a single 4 40GbE QIC be installed in slot 1 and the first two ports be channelized, whereas the remaining two ports are unused. In such a configuration, the native 24 40GbE interfaces are combined with the first 2 40GbE interfaces in the 4 40GbE QIC in slot 1, creating a total of 26×40GE. Each of the 26×40GE interfaces must be channelized into 104 10GbE interfaces. Because the I/O bandwidth is now 1,040 Gbps, the total I/O-to-core bandwidth over-subscription is 13:12. For certain packet sizes, there will be 20 to 30 percent traffic loss, assuming all 104 ports are operating at line rate. Details of the effects of over-subscription are discussed in "Performance" on page 84.

QIC mode

The QIC mode is similar to the 104-port mode, except both QIC slots can be used and there's no requirement to channelize the 40GbE interfaces. However, there are two restrictions:

- The 8 10GbE QIC isn't supported in the QIC mode.
- You cannot channelize the 4 40GbE QIC, only the native 24 40GbE interfaces.

Considering these restrictions, there are two major port configurations:

32 40GbE
: All of the native 24 40GbE interfaces are combined with two 4 40GbE QIC modules for a total of 32 40GbE interfaces on the switch.

96 10GbE and 8 40GbE
: All of the native 24 40GbE interfaces are channelized into 96 10GbE ports, and the two 4 40GbE QICs provided the 8 40GbE interfaces; this is a sneaky port configuration because it stays within the BCM56850 chipset requirement to not exceed 104 total ports.

In summary, the QIC mode turns the Juniper QFX5100-24Q into a 1RU QFX5100-96S or supports 32 40GbE interfaces. Because the I/O bandwidth exceeds the core bandwidth, this system mode is subject to packet loss for certain packet sizes, assuming that all ports are operating at line rate.

Flexible QIC mode

If all of the other system modes weren't enough for you, the Juniper QFX5100-24Q offers yet one final mode: flexible QIC mode. This mode makes it possible for you to

use any type of QIC in the Juniper QFX5100-24Q. There are two restrictions of which you need to be mindful:

- You cannot channelize any of the QICs.
- You cannot channelize ports et-0/0/0 through et-0/0/3 on the Juniper QFX5100-24Q itself, but you can channelize ports et-0/0/4 through et-0/0/23.

Such restrictions create some interesting port configurations, which are presented in Table 3-2.

Table 3-2. QFX5100-24Q flexible QIC mode port configuration options

Native ports	QIC 0	QIC 1	Max 40GbE	Max 10GbE
24 40GbE	4 40GbE	4 40GbE	32 40GbE	80 10GbE 12 40GbE
24 40GbE	8 10GbE	4 40GbE	28 40GbE	88 10GbE 8 40GbE
24 40GbE	8 10GbE	8 10GbE	24 40GbE	96 10GbE 4 40GbE

In summary, with the flexible QIC mode, you can support all of the different types of QIC modules, which most commonly will be deployed as the 32 40GbE configuration when building a spine-and-leaf or Clos IP fabric. Although the overall number of ports can change depending on which QIC you use, it doesn't require a warm reboot as long as you stay in the flexible QIC mode.

Review

The Juniper QFX5100-24Q offers a lot options with respect to port configurations. The general rule of thumb is that the overall number of ports must not exceed 104. There are a total of four system modes and each is unique in the way the switch operates. Table 3-3 summarizes the four system modes and their attributes.

Table 3-3. The Juniper QFX5100-24Q system modes and attributes

Mode	I/O-to-core bandwidth ratio	QIC 0	QIC 1	Max 40GbE	Max 10GbE	Channelize native ports?	Channelize QICs?
Fully subscribed	1:1	No	No	24 40G	96 10GbE	Yes	No
104-port	13:12	Channelize first 2 40GbE	No	None	104 10GbEE	Yes	Channelize first 2 40GbE
QIC	4:3	4 40GbE	4 40GbE	32 40GbE	96 10GbE 8 40GbE	Yes	No

Mode	I/O-to-core bandwidth ratio	QIC 0	QIC 1	Max 40GbE	Max 10GbE	Channelize native ports?	Channelize QICs?
Flexible	4:3	4 40GbE	4 40GbE	32 40GbE	80 10GbE 12 40GbE	Yes	No
Flexible	4:3	8 10GbE	4 40GbE	28 40GbE	88 10GbE 8 40GbE	Yes	No
Flexible	7:6	8 10GbE	8 10GbE	24×40GbE	96 10GbE 4 40GbE	Yes	No

It's important to consider what role within the architecture the Juniper QFX5100-24Q fits. Depending on the system mode, it can fit into any number of possibilities. For example, in QIC mode, the Juniper QFX5100-24Q supports 32 40GbE interfaces, which makes a lot sense in the core and aggregation of a network. On the other hand, running the Juniper QFX5100-24Q in 104-port mode offers 104 10GbE interfaces in a 1RU form factor, which makes a lot of sense in the access tier of the network. The Juniper QFX5100 series has been designed from the ground up to give you more options.

Performance

With the critical design considerations out of the way, it's now time to focus on the performance characteristics of the Juniper QFX5100 series. Previously in this chapter, we explored the BCM56850 chipset and how the I/O and core bandwidth work together in a balancing act of port density versus performance. Performance can be portrayed through two major measurements: throughput and latency. Let's examine each of them.

Throughput

The throughput Juniper QFX5100 switches will vary depending on system mode in which the device is operating. The fully subscribed (default) mode has an over-subscription of 1:1 and doesn't have any loss in traffic when all of the ports are operating at line rate. All of the other modes will have some level of I/O and core bandwidth over-subscription (refer to Table 3-3).

The key questions are the following:

- What conditions cause over-subscription?
- What packet sizes are affected?
- How much traffic is dropped?

To over-subscribe the switch, it must be currently processing more traffic than the core bandwidth can handle, which is 960 Gbps. The best way to answer the rest of the questions is with the graph shown in Figure 3-4.

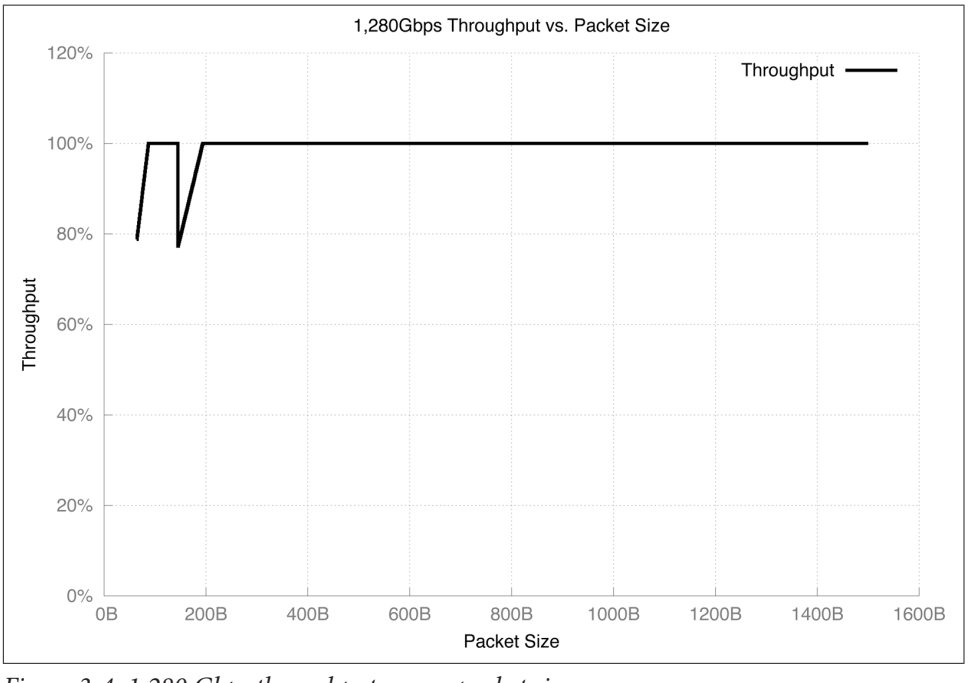


Figure 3-4. 1,280 Gbps throughput versus packet size

There's a lot happening in the graph in Figure 3-4. It can be summarized as the following:

- Packet sizes 64B through 86B vary in performance 78 to 99 percent.
- Packet sizes 87B through 144B offer line-rate performance.
- Packet sizes 145B through 193B vary in performance 77 to 99 percent.
- Packet sizes 194B through 12,288B offer line-rate performance.

In summary, only packet sizes between 64B through 86B and 145B through 193B have varying traffic loss of 20 to 1 percent when there is congestion on the switch. Another way to view it is out of 12,228 possible packet sizes, only 0.005 percent suffer traffic loss. If you want to be pedantic and assume only 1,514 possible packet sizes, only 0.05 percent suffer traffic loss.

The reason the chipset is able to forward some packet sizes at line rate and not others is how the stepping in line-rate frequency is required to process some packet sizes versus others. Packet sizes ranging from 64B to 86B and 145B to 193B require a

higher frequency to process than other sizes and are subject to a varying amount of traffic loss during switch congestion.

Keep in mind that traffic loss is only experienced in system modes other than fully subscribed/default.

Latency

Latency is the measurement of time between when a packet enters the switch on an ingress port and when it leaves the switch on an egress port, as illustrated in Figure 3-5.

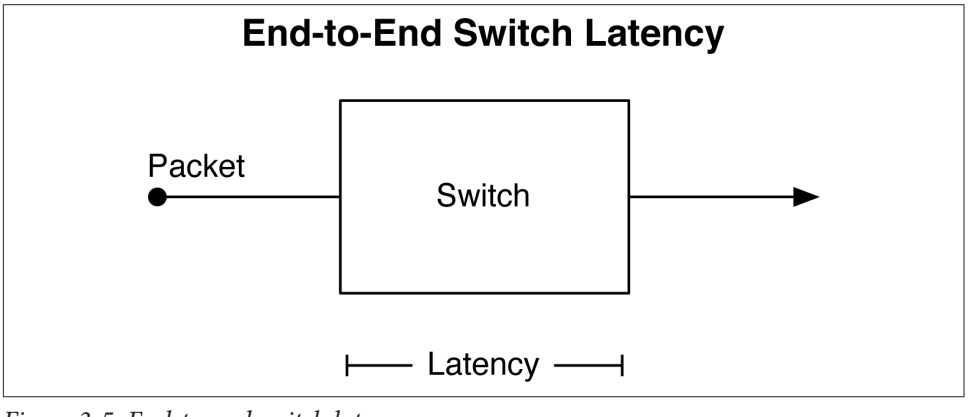


Figure 3-5. End-to-end switch latency

With modern hardware such as the Juniper QFX5100 series, the amount of latency continues to decrease. In the vast majority of use cases, latency isn't a major concern; however, there exists a subsegment in the financial-services markets and high-performance computing that specialize in low latency.

Cut-through and store-and-forward

There are two modes that greatly impact the switch's overall latency: cut-through and store-and-forward. Each mode is purposely designed to excel in specific use-cases.

Cut-Through
: A switch that operates in a cut-through mode will begin to transmit the packet on the egress port at the same time it is receiving it on the ingress port. The benefit here is a reduction in overall latency within the switch because there's no delay in transmitting the packet to its destination. The drawback is that cut-through mode has no way of discarding a corrupt packet, because the majority of the

packet will already be transmitted on the egress port before the FCS is received on the ingress port. In larger networks or with multicast, cut-through mode can cause a lot of unnecessary processing in upstream devices when replicating corrupt packets.

Store-and-Forward
: The default setting for the Juniper QFX5100 family is store-and-forward; this mode is how most switches have operated for a long time. The ingress packet must be fully received before the switch will transmit the packet on the egress port. The advantage is that the switch can perform error checks on the packet and discard it if it's corrupt. The drawback is that store-and-forward requires a buffer within the switch to store the packet while it's being received; this increases the cost and overall latency.

Unless you're building a financial trading platform or high-performance computing environment, the default mode of store-and-forward will generally meet and exceed all of your latency requirements.

Conditions for cut-through

By default, the Juniper QFX5100 family operates in store-and-forward mode. To enable cut-through mode, you must issue and commit the following command:

```
[edit]
dhanks@QFX5100# set forwarding-options cut-through
```

Don't be fooled: this command is just the first step to enable cut-through mode. There are many conditions that a packet must meet in order to be eligible for cut-through mode; otherwise, it defaults back to store-and-forward. This decision process is done on a per-packet basis, although the cut-through is a system-wide setting. The first set of requirements is that only matching ingress and egress interface speeds are eligible for cut-through mode, as presented in Table 3-4.

Table 3-4. Forwarding modes based on port speed and system mode

Ingress port	Egress port	Cut-through (CT) system mode	Store-and-forward (SF) system mode
10GbE	10GbE	CT	SF
40GbE	40GbE	CT	SF
10GbE	40GbE	SF	SF
40GbE	10GbE	SF	SF
1GbE	1GbE	CT	SF
1GbE	10GbE	SF	SF
10GbE	1GbE	SF	SF

For example, if the Juniper QFX5100 switch were configured to be in cut-through mode, but a packet arrived on a 40GbE ingress interface and was transmitted on a 10GbE egress interface, that packet would not be eligible for cut-through mode and would default back to store-and-forward.

If the packet meets the conditions specified in Table 3-4, it will be subject to additional conditions before being forwarded via cut-through.

- The packet must not be destined to the routing engine.
- The egress port must have an empty queue with no packets waiting to be transmitted.
- The egress port must not have any shapers or rate limiting applied.
- The ingress port must be in-profile if it's subject to rate limiting.
- For multicast packets, each egress port must meet all conditions. If one egress port out of the set doesn't meet the conditions, all multicast packets will be transmitted via store-and-forward; the chipset doesn't support partial cut-through packets.

To further understand the benefits of improved latency of cut-through mode, let's compare it directly to store-and-forward with different sized packets up to 1,514 bytes, as illustrated in Figure 3-6.

The cut-through latency increases slowly from 64 bytes up to about 600 bytes and remains steady at about 0.73 µs. On the other hand, the store-and-forward is fairly linear from 64 bytes all the way to 1,514 bytes. In summary, cut-through and store-and-forward have less than 1 µs of latency when the packet is less than 1,514 bytes.

Let's take a look at what happens when you enable jumbo frames. Figure 3-7 starts in the same place at 64 bytes but goes all the way up to 9,216 bytes.

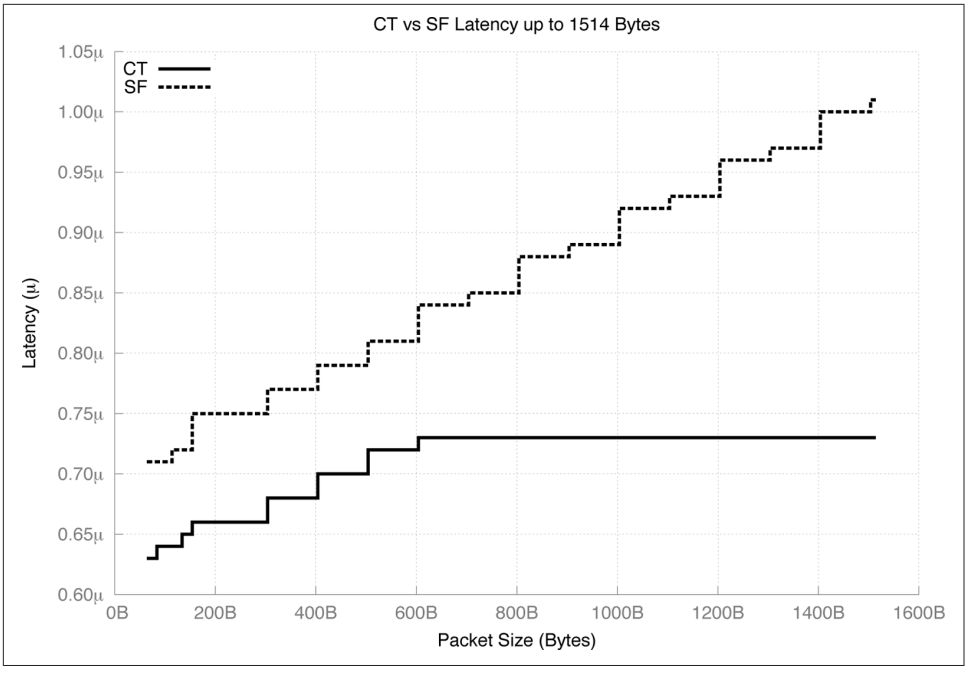


Figure 3-6. Approximate latency for the BCM56850 chipset using 40GbE with frames up to 1,514 bytes

In summary, the store-and-forward continues to stay fairly linear from 64 bytes to 9,216 bytes; however cut-through flattens out at approximately 0.73 µs from 600 bytes to 9,216 bytes. Store-and-forward follows a linear progression simply because the latency is a factor of how large the packet is. The larger the packet, the more memory it takes to buffer it before it's allowed to be transmitted. Cut-through mode stays flat because it simply begins transmitting the packet as soon as it's received; thus the packet size is never a factor in the overall latency.

These graphs represent approximate latency on the BCM56850 chipset using 40GbE interfaces. Actual values will vary based on firmware, port speed, and other factors. If latency is critical to your environment, you need to evaluate the latency in your lab under controlled conditions.

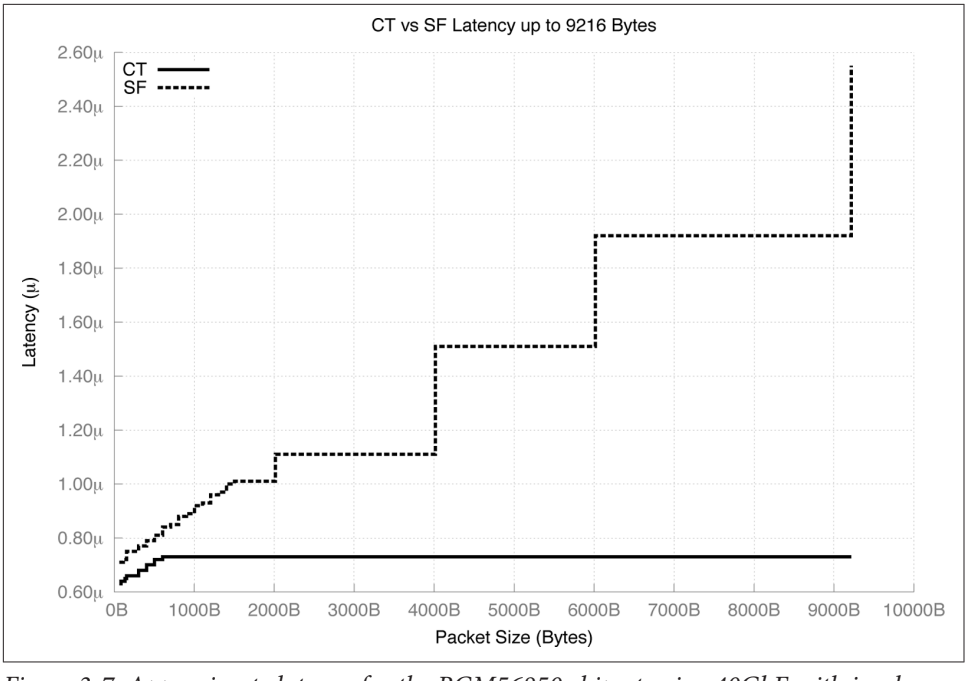


Figure 3-7. Approximate latency for the BCM56850 chipset using 40GbE with jumbo frames

Scale

Scale can be expressed many different ways. The most common methods are the configuration maximums of the control plane and data plane. It's also common to peg the scaling maximums to the OSI model, for example Layer 2 versus Layer 3. The Juniper QFX5100 series is unique in the sense that you can adjust the balance of Layer 2 versus Layer 3 data plane scale. Let's dive into the details.

Unified Forwarding Table

The Juniper QFX5100 series has the unique ability to use a customized forwarding table. The forwarding table is broken into three major tables:

MAC Address Table
: In a Layer 2 environment, the switch will learn new MAC addresses and it stores them in the MAC address table.

Layer 3 Host Table
: In a Layer 2 and Layer 3 environment, the switch will also learn which IP addresses are mapped to which MAC addresses; these key-value pairs are stored in the Layer 3 host table.

Longest Prefix Match (LPM) Table

In a Layer 3 environment, the switch will have a routing table, and the most specific route will have an entry in the forwarding table to associate a prefix/netmask to a next-hop; this is stored in the LPM table. The one caveat is that all IPv4 /32 prefixes and IPv6 /128 prefixes are stored in the Layer 3 host table.

Traditionally, these tables have been statically defined from the vendor and only support a fixed number of entries, which ultimately limits what role in the architecture into which a traditional switch can fit.

The Unified Forwarding Table (UFT) in the Juniper QFX5100 family allows you to dynamically move around forwarding table resources so that you can tailor the switch to your network. In summary, the UFT offers five preconfigured profiles from heavy Layer 2 to heavy Layer 3 allocations, as shown in Table 3-5.

Table 3-5. The Juniper QFX5100 UFT profiles

Profile	MAC addresses	L3 hosts	LPM
`l2-profile-one`	288,000	16,000	16,000
`l2-profile-two`	224,000	56,000	16,000
`l3-profile-three`	160,000	88,000	16,000
`l3-profile`	96,000	120,000	16,000
`lpm-profile`	32,000	16,000	128,000

The UFT is a very powerful tool that completely changes the personality of the switching, allowing it to move freely throughout the network architecture. Each profile has a linear progression toward a larger Layer 3 host table, as depicted in Figure 3-8.

Using a heavy MAC address table makes it possible for Juniper QFX5100 switches to handle a lot of Layer 2 traffic such as a traditional virtualization environment with servers hosting a large amount of VMs. The last profile gives you the ability to operate Juniper QFX5100 devices in the core of a network architecture or use them as a building block in a large Clos IP fabric; this is because an IP fabric by nature will have a larger routing table than MAC address tables.

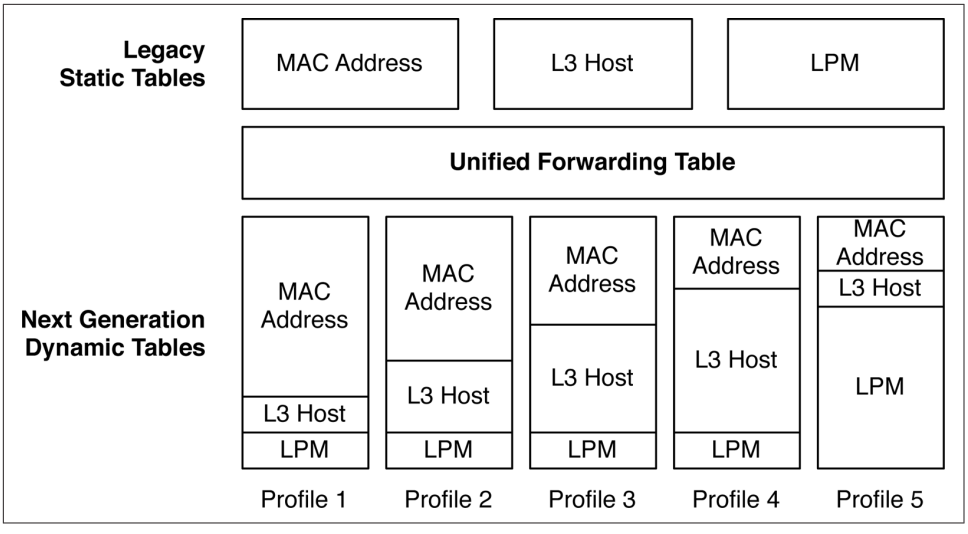


Figure 3-8. Juniper QFX5100 series UFT

To check the current forwarding mode the Juniper QFX5100 switch, use the **`show chassis forwarding-options`** command:

```
dhanks@qfx5100> show chassis forwarding-options
------------------------------------------------------------------------
Current UFT Configuration:
l2-profile-three
```

You can see from the preceding output that this particular Juniper QFX5100 switch is currently in `l2-profile-three mode`, which gives the forwarding table 160K MAC addresses, 88K L3 hosts, and 16K LPM entries. The forwarding table can be changed by using the following command:

```
[edit]
dhanks@qfx5100# set chassis forwarding-options ?
Possible completions:
+ apply-groups         Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
  l2-profile-one       MAC: 288K L3-host: 16K LPM: 16K. This will restart PFE
  l2-profile-three     MAC: 160K L3-host: 144K LPM: 16K. This will restart PFE
  l2-profile-two       MAC: 224K L3-host: 80K LPM: 16K. This will restart PFE
  l3-profile           MAC: 96K L3-host: 208K LPM: 16K. This will restart PFE
  lpm-profile          MAC: 32K L3-host: 16K LPM: 128K. This will restart PFE
```

Be mindful that when you change the UFT profile and commit, the BCM56850 chipset will need to perform a warm reboot, and there will be temporary traffic loss.

Hashing

The Juniper QFX5100 uses a sophisticated hashing algorithm called RTAG7 to determine the next-hop interface for Equal-Cost Multipath (ECMP) routing and Link Aggregation (LAG). Each packet is subject to the following fields when determining the next-hop interface:

- Source MAC address
- Destination MAC address
- Ethernet type
- VLAN ID
- Source IP address
- Destination IP address
- IPv4 protocol or IPv6 next header
- Layer 4 source port
- Layer 4 destination port
- MPLS label

There are also two additional fields that are used to calculate the hash that are internal to the system:

- Source device ID
- Source port ID

The following types of protocols are supported for ECMP on the Juniper QFX5100 as of Junos 13.2X51-D20.2:

- IPv4
- IPv6
- MPLS
- MAC-in-MAC

Note that additional protocols can be supported with a new software release; please check the release notes for Junos going forward.

The hash algorithm for ECMP and LAG use the same packet fields as those just listed, but note that an internal hash index is calculated differently. This method avoids traffic polarization when a LAG bundle is part of an ECMP next-hop.

Resilient Hashing

One of the challenges in the data center when building IP fabrics with stateful devices —such as firewalls—is minimizing the number of next-hop changes during link failures. For example, the Juniper QFX5100 will perform standard RTAG7 hashing on all ingress flows and send out a next-hop as dictated by the hashing algorithm. If a firewall were to fail, the standard RTAG7 hashing algorithm on the QFX5100 switch would be impacted and the egress next-hop for new and existing flows would be assigned next-hops. The end result is that existing flows would be hashed to a new firewall. Because the new firewall doesn't have a session entry for the rerouted flow, the firewall would simply discard the traffic, as shown in Figure 3-9.

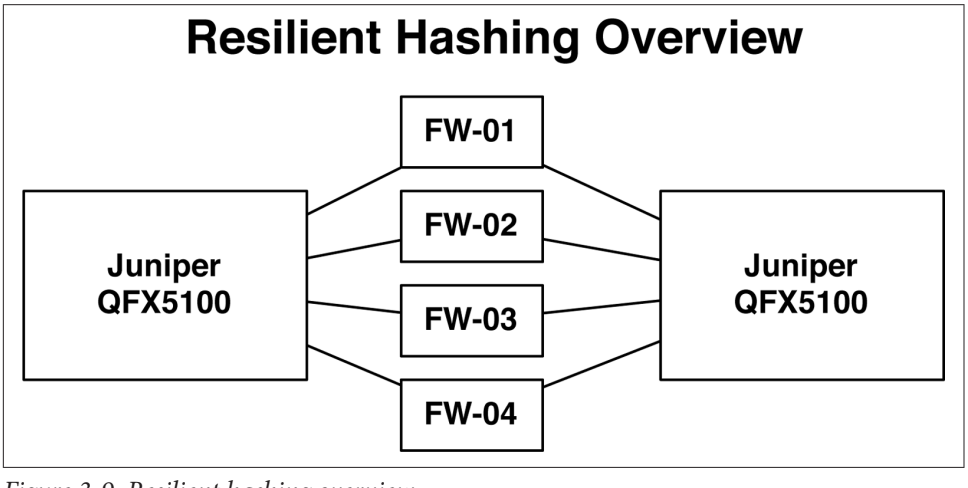


Figure 3-9. Resilient hashing overview

The Juniper QFX5100 supports a new type of hashing called *resilient hashing* that minimizes the number of next-hop changes during link failures. If a firewall were to fail, the Juniper QFX5100 would keep the existing flows mapped to their existing egress next-hops. The end result is that when a firewall fails, all of the other flows continue to flow through their existing firewalls without impact.

The Juniper QFX5100 series also supports resilient hashing for a LAG interface, as well. In summary, resilient hashing supports both Layer 3 ECMP and LAG ECMP.

Layer 2 LAG

To enable resilient hashing for Layer 2 LAG members, use the following command (replace `ae0` with the intended interface name for your environment):

```
# set interface ae0 aggregated-ether-options resilient-hash
```

Layer 3 ECMP

To enable resilient hashing for Layer 3 ECMP, use the following command:

```
# set forwarding-options enanced-hash-key ecmp-resilient hash
```

Configuration Maximums

The Juniper QFX5100 has a set of configuration maximums that you need to be aware of as you design your network. The Juniper QFX5100 should work just fine in the majority of use cases, but there could be instances for which you might need more scale. Use Table 3-6 as a reference.

Table 3-6. QFX5100 family configuration maximums

Key	Value
MAC addresses	288 K (UFT l2-profile-one)
ARP entries	48 K
Jumbo frame size	9,216 bytes
IPv4 unicast routes	128 K prefixes, 208 K host routes
IPv4 multicast routes	104 K
IPv6 unicast routes	64 K
IPv6 multicast routes	52 K
VLAN IDs	4,094
FCoE VLANs	4,094
Link aggregation groups	128
Members per LAG	32
Firewall filters	4 K
ECMP	64
MSTP instances	64
VSTP instances	253
Mirroring destination ports	4
Mirroring sessions	4
Mirroring destination VLANs	4

There will be some configuration maximums such as the UFT, MAC addresses, and others that are pinned to the BCM 56850 chipset and can never be increased. However there are other configuration maximums such as ECMP, link aggregation groups, and STP instances that you can increase over time with Junos software updates.

Summary

This chapter covered many of the design considerations that you must take into account before looking at the scale of each role in the architecture. These design considerations are using compute virtualization in the data center and an overlay architecture. Moving to an overlay architecture in the data center changes many of the traditional scaling requirements with which you are familiar.

The Juniper QFX5100-24Q has four different system modes to handle over-subscription to provide a customized personality depending on the use case. The system modes are:

- Fully subscribed mode (default)
- 104-port mode
- QIC mode
- Flexible QIC mode

Each of the system modes impact how the I/O and core bandwidth are handled, ultimately changing the throughput characteristics of the switch.

The Juniper QFX5100 chipset also has a next generation UFT with which you can choose one of five preconfigured profiles from Layer 2 heavy to Layer 3 heavy; this gives you the freedom to place the Juniper QFX5100 switch anywhere in your network and fine-tune the logical scale to match its role in the network.

Many factors impact the latency of a network switch. The Juniper QFX5100 family offers two forwarding modes: cut-through and store-and-forward. Cut-through gives you the lowest possible latency at the expense of forwarding corrupt frames. Store-and-forward has slightly higher latency, but completely buffers the packet and is able to discard corrupt packets.

In summary, the Juniper QFX5100 family gives you the power of options. When trying to solve complicated problems, the easiest method is to break it down into simple building blocks. The more options that are available to you, the greater your chances are of executing a successful data center strategy and architecture. Let's review the options the Juniper QFX5100 series affords you to consider in this chapter:

- Traditional IP network versus overlay architecture
- VMware NSX versus Juniper Contrail
- Four system modes to fine-tune the over-subscription in the data plane
- Five profiles to fine-tune the logical scaling in the data plane
- Cut-through mode versus store-and-forward mode

Juniper QFX5100 switches are very exciting, and, as of this writing, represent Juniper's best switches ever created. As you work your way through this book, think about all of the different places in your network where the Juniper QFX5100 series of switches could be used and make it better.

Chapter Review Questions

1. Which overlay control plane protocols does the Juniper QFX5100 family support?

a. Open vSwitch Database

b. Device Management Interface

c. All of the above

d. None of the above

2. How does the Juniper QFX5100 series support bare-metal servers in an overlay architecture?

a. Forward all traffic from the bare-metal server to the SDN controller

b. Forward all traffic from the bare-metal server to the closest hypervisor VTEP

c. Handle all encapsulation and forwarding in the switch's hardware

d. Implement a VTEP inside of the switch with a control plane protocol

3. What's the core bandwidth of the BCM56850 chipset?

a. 1,280 Gbps

b. 960 Gbps

c. 720 Gbps

d. 480 Gbps

4. How many system modes does the Juniper QFX5100-24Q have?

a. 2

b. 3

c. 4

d. 5

5. What's the I/O bandwidth to core bandwidth ratio of the Juniper QFX5100-24Q when using 32 40GbE interfaces?

a. 1:1

b. 13:12

c. 4:3

d. 5:4

6. How many preconfigured profiles are in the Juniper QFX5100 UFT?

a. 1

b. 3

c. 5

d. 7

7. What's the maximum number of MAC addresses in a Juniper QFX5100 switch?

a. 128K

b. 224K

c. 256K

d. 288K

8. What's the maximum size of an Ethernet frame in the Juniper QFX5100 series?

a. 2,048 bytes

b. 4,000 bytes

c. 8,192 bytes

d. 9,216 bytes

Chapter Review Answers

*1. **Answer: C.** Juniper QFX5100 series switches support both OVSDB and DMI control plane protocols.*

*2. **Answer: C and D.** Trick question. The Juniper QFX5100 family handles the data plane encapsulation in hardware and creates a VTEP inside of the switch for MAC address learning and service provisioning.*

*3. **Answer: B.** Juniper QFX5100 switches use the BCM56850 chipset, which has a core bandwidth of 960 Gbps and I/O bandwidth of 1,280 Gbps.*

*4. **Answer: C.** The Juniper QFX5100-24Q has four system modes: (1) fully subscribed, (2) 104 port, (3) QIC, and (4) flexible QIC.*

*5. **Answer: C.** 32 40GbE interfaces requires 1,280 Gbps of I/O bandwidth, which creates a 4:3 ratio of I/O bandwidth to core bandwidth.*

*6. **Answer: C.** Juniper QFX5100 switches support five UFT profiles: (1) l2-profile-one (2) l2-profile-two (3) l2-profile-three (4) l3-profile, and (5) lpm-profile.*

*7. **Answer: D.** The Juniper QFX5100 family supports up to 288K MAC addresses in UFT l2-profile-one.*

*8. **Answer: D.** The Juniper QFX5100 series supports jumbo frames up to 9,216 bytes.*

CHAPTER 4

One Box, Many Options

It isn't often that you can accurately predict all of the business demands and requirements when working in IT; it seems as if the target is always moving, requiring that you always adjust. Adapting to change is required when working with technology and supporting constantly changing business requirements. To react quickly, you must have the flexibility to choose the best tool for the job. Having multiple options is always a winning strategy against any opponent.

The Juniper QFX5100 is a powerful series of switches because they give you the power inherent in having many different options at hand. You are not forced to use a particular option but instead are empowered to make your own determination as to what technology option makes the most sense in a particular situation. The Juniper QFX5100 family can support the following technology options:

- Standalone
- Virtual Chassis Fabric (VCF)
- QFabric Node
- Virtual Chassis
- Multi-Chassis Link Aggregation (MC-LAG)
- Clos Fabric

Not only do you have multiple options, but you can also choose to deploy a Juniper architecture or an open architecture (see Figure 4-1). You have the ability to take advantage of turnkey Ethernet fabrics or simply create your own and integrate products from other vendors as you go along.

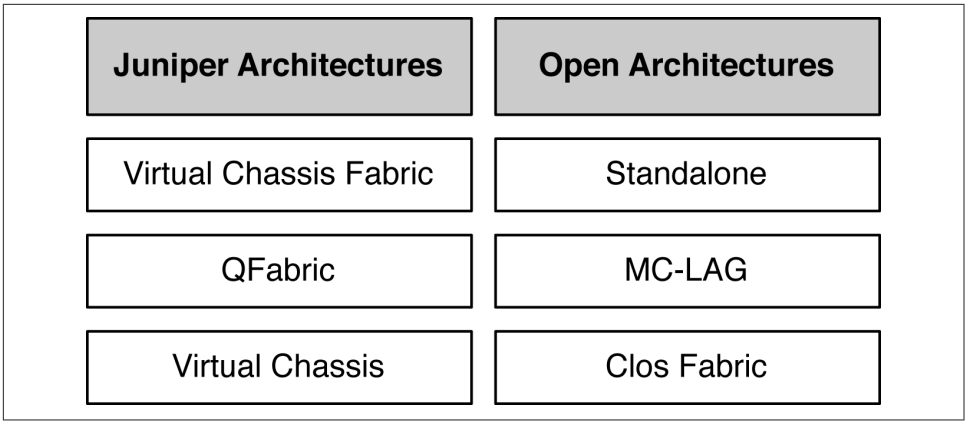


Figure 4-1. Juniper architectures and open architectures options

This chapter is intended to introduce you to the many different options the Juniper QFX5100 offers. We'll investigate each option, one by one, and get a better idea about what each technology can do for you and where it can be used in your network.

Standalone

The most obvious way to implement a Juniper QFX5100 switch is in standalone mode, just a simple core, aggregation, or access switch. Each Juniper QFX5100 switch operates independently and uses standard routing and switching protocols to forward traffic in the network, as illustrated in Figure 4-2.

The Juniper QFX5100 switches in the core layer in Figure 4-2 are running only Open Shortest Path First (OSPF) to provide Layer 3 connectivity. The switches in the aggregation layer are running both OSPF and Virtual Router Redundancy Protocol (VRRP); this is to provide Layer 3 connectivity to both the core and access layers. The links from the aggregation switches to the access switch are simple Layer 2 interfaces running IEEE 802.1Q. The aggregation switch on the left is the VRRP master. It provides Layer 3 gateway services to the access switch.

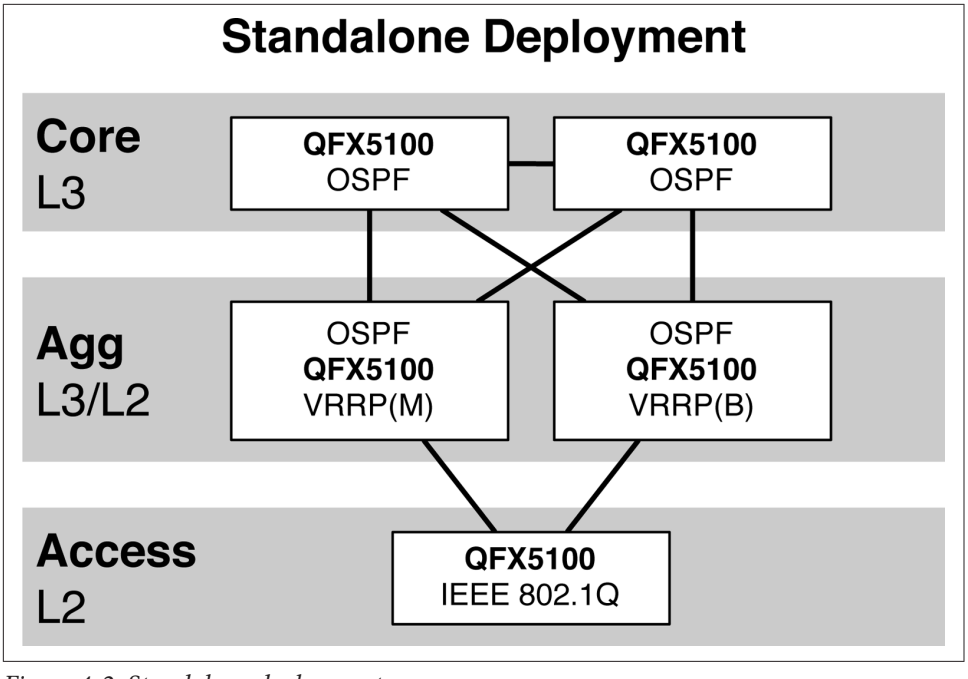


Figure 4-2. Standalone deployment

In Figure 4-2, notice that two aggregation switches do not have a connection between them. This is intentional. The VRRP protocol requires a Layer 2 connection between master and backup switches; otherwise, the election process wouldn't work. In this example, the two switches have a Layer 2 connection through the access switch and VRRP is able to elect a master. Another design benefit from removing the Layer 2 link between the aggregation switches is that it physically eliminates the possibility of a Layer 2 loop in the network.

The benefit of a standalone deployment is that you can easily implement the Juniper QFX5100 switch into an existing network using standards-based protocols. Easy peasy!

Virtual Chassis

When Juniper released its first switch, the EX4200, one of the innovations was Virtual Chassis, which took traditional "stacking" to the next level. By virtualizing all of the functions of a physical chassis, this technology made it possible for a set of physical switches to form a virtualized chassis, complete with master and backup routing engines, line cards, and a true single point of management.

The Juniper QFX5100 family continues to support Virtual Chassis. You can form a Virtual Chassis between a set of QFX5100 switches or create a mixed Virtual Chassis by using the QFX3500, QFX3600, or EX4300, as demonstrated in Figure 4-3.

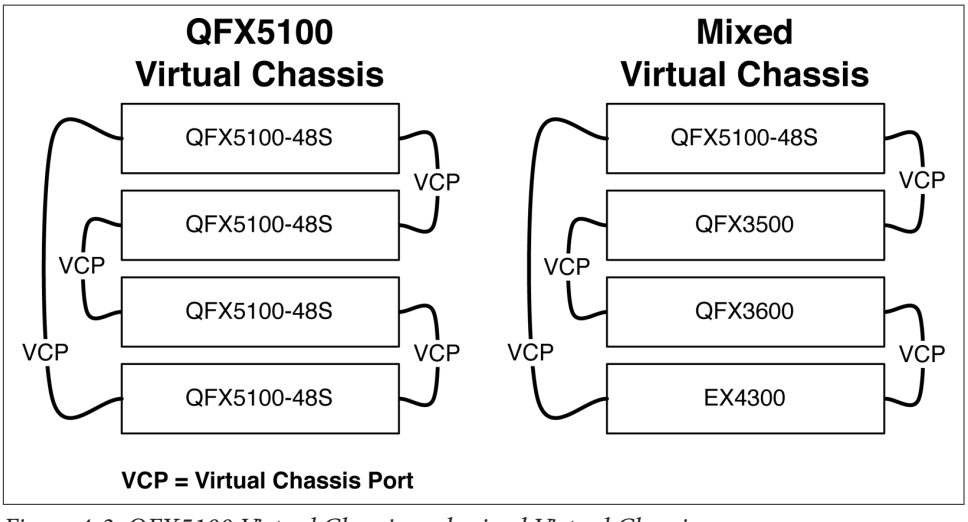


Figure 4-3. QFX5100 Virtual Chassis and mixed Virtual Chassis

Figure 4-3 doesn't specify the best current practice on how to cable VCPs between switches; instead, it simply illustrates that the Juniper QFX5100 series supports regular Virtual Chassis and a mixed Virtual Chassis with other devices.

Virtual Chassis is a great technology to reduce the number of devices to manage in the access tier of a data center or campus. In the example in Figure 4-3, the Juniper QFX5100 Virtual Chassis has four physical devices, but only a single point of management.

One drawback of Virtual Chassis is the scale and topology. Virtual Chassis allows a maximum of 10 switches and is generally deployed in a ring topology. Traffic going from one switch to another in a ring topology is subject to nondeterministic latency and over-subscription, depending on how many transit switches are between the source and destination. To be able to take innovation to the next level, a new technology is required to increase the scale and provide deterministic latency and over-subscription.

For more information about Virtual Chassis check out *JUNOS Enterprise Switching* by Doug Marschke and Harry Reynolds (O'Reilly).

QFabric

QFabric is the next step up from Virtual Chassis. It's able to scale up to 128 switches and uses an internal 3-stage Clos topology to provide deterministic latency and oversubscription. With higher scale and performance, QFabric has the ability to collapse the core, aggregation, and access into a single data center tier, as shown in Figure 4-4.

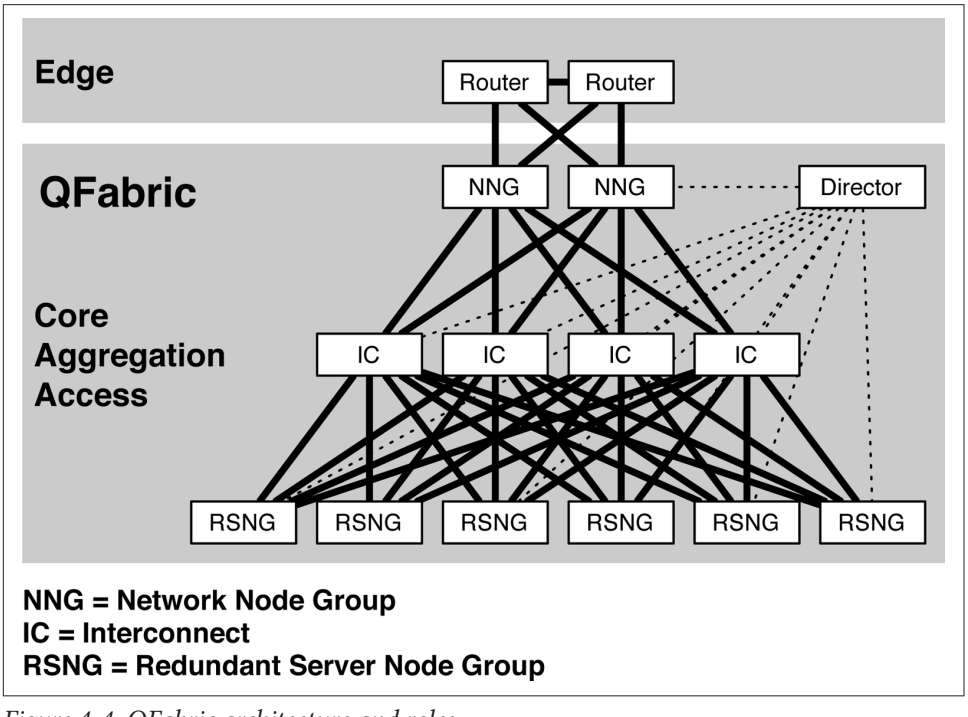


Figure 4-4. QFabric architecture and roles

All of the components in the core, aggregation, and access tier (the large gray box in Figure 4-4) make up the QFabric architecture. The components and function of a QFabric architecture are listed in Table 4-1.

Table 4-1. QFabric architecture components, tiers, and functions

Component	Tier	Function
IC	Core and aggregation	All traffic flows through the IC switches; it acts as the middle stage in a 3-stage Clos fabric.
RSNG	Access	All servers, storage, and other end points connect into the Redundant Server Node Group (RSNG) top-of-rack (ToR) switches for connectivity into the fabric.
NNG	Routing	Any other devices that need to peer to QFabric through a standard routing protocol such as OSPF or Border Gateway Protocol (BGP) are required to peer into a Network Node Group (NNG).

Component	Tier	Function
Director	Control plane	Although QFabric is a set of many physical devices, it's managed as a single switch. The control plane has been virtualized and placed outside of the fabric. Each component in QFabric has a connection to the Director. All configuration and management is performed from a pair of Directors.

Managing an entire data center network through a single, logical switch has tremendous operational benefits. You no longer need to worry about routing and switching protocols between the core, aggregation, and access tiers in the network. The QFabric architecture handles all of the routing and switching logic for you; it simply provides you a turnkey Ethernet fabric that can scale up to 128 ToR switches.

The Juniper QFX5100 series is able to participate in the QFabric architecture as a ToR switch or RSNG. A important benefit to using a Juniper QFX5100 switch as an RSNG in a QFabric architecture is that it increases the logical scale of QFabric as compared to using the QFX3500 or QFX3600 as an RSNG. A QFabric data center only using QFX5100 RSNGs can reach logical scaling, which is described in Chapter 3.

Virtual Chassis Fabric

If the scale of Virtual Chassis is a bit too small and the QFabric a bit too big, Juniper's next innovation is VCF; it's a perfect fit between traditional Virtual Chassis and QFabric. By adopting the best attributes of Virtual Chassis and QFabric, Juniper has created a new technology with which you can build a plug-and-play Ethernet fabric that scales up to 32 members and provides deterministic latency and over-subscription with an internal 3-stage Clos topology, as depicted in Figure 4-5.

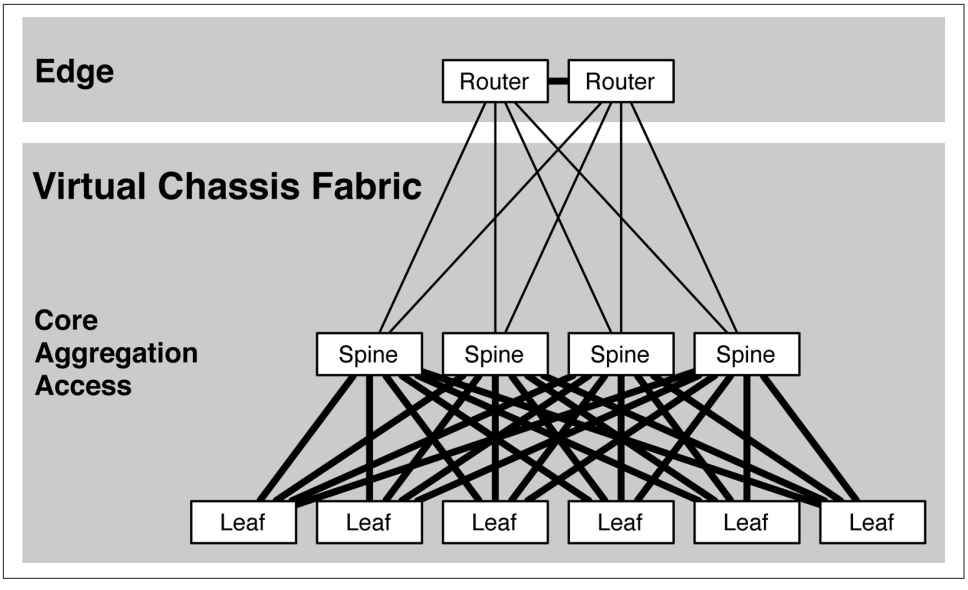


Figure 4-5. VCF architecture

At first glance, VCF and QFabric look very similar. A common question is, "What's different?" Table 4-2 looks at what the technologies have in common and what separates them.

Table 4-2. Comparison of QFabric and VCF

Attribute	QFabric (QFX3000-G)	VCF
Physical scale	128 nodes	32 nodes
Control plane connectivity	Out-of-band	In-band
Connectivity	Routers must connect to NNGs. Hosts must connect to RSNGs. Only NNGs or RSNGs can connect to ICs.	Universal Ports. Any port on any switch can support any host and protocol. No limitations.
Plug-and-play	No. Requires external cabling and minimal configuration	Yes
Software upgrades	NSSU	ISSU
ECMP	Yes	Yes
Full Layer 2 and Layer 3	Yes	Yes
Lossless Ethernet/DCB	Yes	Yes
Universal Server Ports	No	Yes

VCF offers features and capabilities that are above and beyond QFabric and is a great technology to collapse multiple tiers in a data center network. As of this writing, the only limitation is that VCF allows a maximum of 32 members. One of the main dif-

ferences is the introduction of a concept called *Universal Server Ports*. This makes it possible for a server to plug into any place into the topology. For example, a server can plug into either a leaf or spine switch in VCF. On the other hand, with QFabric you can plug servers only into QFabric Nodes, because the IC switches are reserved only for QFabric nodes.

The Juniper QFX5100 family can be used in both the spine and leaf roles of VCF. You can use the EX4300 series in VCF, too, but only as a leaf. Table 4-3 presents device compatibility in a VCF as of this writing.

Table 4-3. VCF compatibility

Switch	Spine	Leaf
QFX5100-24Q	Yes	Yes
QFX5100-96S	Yes	Yes
QFX5100-48S	Yes	Yes
QFX5100-48T	No	Yes
QFX3500	No	Yes
QFX3600	No	Yes
EX4300	No	Yes

In summary, the Juniper QFX5100 series must be the spine in a VCF, but you can use all of the other QFX5100 models, as well as QFX3500, QFX3600, and EX4300 series switches as a leaf.

MC-LAG

Virtual Chassis, QFabric, and VCF are all Juniper architectures. Let's move back into the realm of open architectures and take a look at MC-LAG. In a network with multiple vendors, it's desirable to choose protocols that support different vendors, as shown in Figure 4-6.

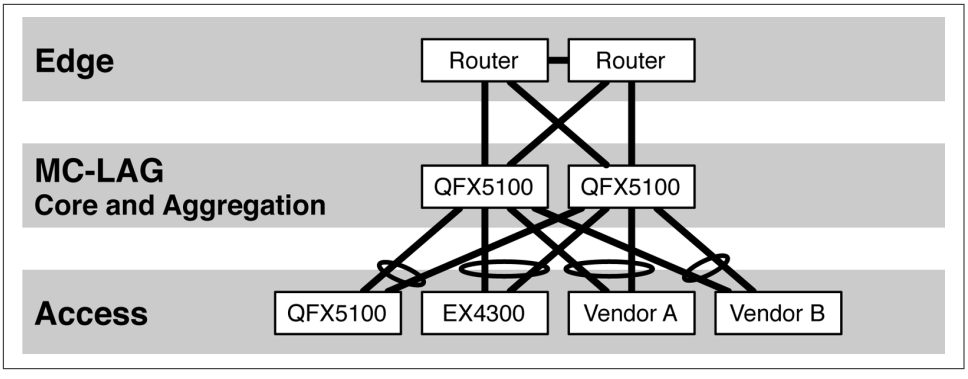


Figure 4-6. MC-LAG architecture

The figure shows that the Juniper QFX5100 family supports the MC-LAG protocol between two switches. All switches in the access tier simply speak IEEE 802.1AX/LACP to the pair of QFX5100 switches in the core and aggregation tier. From the perspective of any access switch, it's unaware of MC-LAG and only speaks IEEE 802.1AX/LACP. Although there are two physical QFX5100 switches running MC-LAG, the access switches only see two physical interfaces and combine them into a single logical aggregated Ethernet interface.

All of the Juniper QFX5100 platforms support MC-LAG, and you can use any access switch in the access layer that supports IEEE 802.3ad/LACP.

For more information about MC-LAG, check out *Juniper MX Series* by Douglas Richard Hanks, Jr. and Harry Reynolds (O'Reilly).

Clos Fabric

When scale is a large factor in building a data center, many engineers turn toward building a Clos fabric with which they can easily scale to 100,000s of ports. The most common Clos network is a 3-stage topology, as illustrated in Figure 4-7.

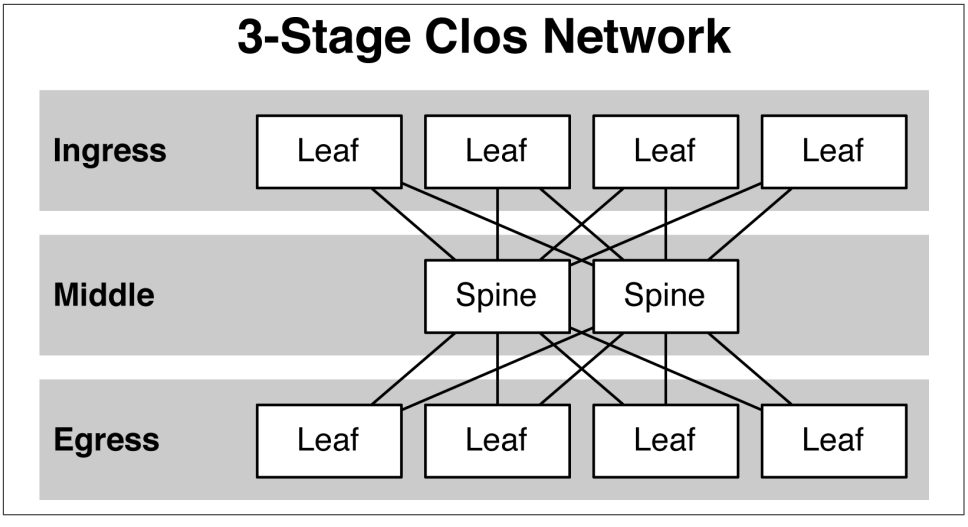


Figure 4-7. Architecture of Clos network

Depending on the port density of the switches used in a Clos network, the number of leaves can easily exceed 500 devices. Due to the large scale of Clos networks, it's always a bad idea to use traditional Layer 2 protocols such as spanning tree or MC-LAG because it creates large broadcast domains and excessive flooding. Clos fabrics are Layer 3 in nature because routing protocols scale in an orderly fashion and reduce the amount of flooding. If Layer 2 connectivity is required, using higher level architectures such as overlay networking go hand-in-hand with Clos networks. There are many options when it comes to routing protocols, but traditionally, BGP is used primarily for three reasons:

- Support multiple protocols families (inet, inet6, evpn)
- Multivendor stability
- Scale
- Traffic engineering and tagging

The Juniper QFX5100 series works exceedingly well at any tier in a Clos network. The Juniper QFX5100-24Q works well in the spine because of its high density of 40GbE interfaces. Other models such as the Juniper QFX5100-48S or QFX5100-96S work very well in the leaf because most hosts require 10GbE access, and the spine operates at 40GbE.

Clos fabrics are covered in much more detail in Chapter 7.

Transport Gymnastics

The Juniper QFX5100 series handles a large variety of different data plane encapsulations and technologies. The end result is that a single platform can solve many types of problems in the data center, campus, and WAN. There are five major types of transport that Juniper QFX5100 platforms support:

- MPLS
- VXLAN
- Ethernet
- FCoE
- HiGig2

The Juniper QFX5100 is pretty unique in the world of merchant silicon switches because of the amount of transport encapsulations enabled on the switch. Typically other vendors don't support MPLS, Fibre Channel over Ethernet (FCoE), or HiGig2. Now that you have access to all of these major encapsulations, what can you do with them?

MPLS

Right out of the box, MPLS is one of the key differentiators of Juniper QFX5100 switches. Typically, such technology is reserved only for big service provider routers such as the Juniper MX. As of this writing, the QFX5100 family supports the following MPLS features:

- LDP
- RSVP
- LDP tunneling over RSVP
- L3VPN
- MPLS automatic bandwidth allocation
- Policer actions
- Traffic engineering extensions for OSPF and IS-IS
- MPLS Ping

One thing to note is that Juniper QFX5100 platforms don't support as many MPLS features as the Juniper MX, but all of the basic functionality is there. The Juniper QFX5100 family also supports MPLS within the scale of the underlying Broadcom chipset, as outlined in Chapter 3.

Virtual Extensible LAN

The cool kid on the block when it comes to data center overlays is Virtual Extensible LAN (VXLAN). By encapsulating Layer 2 traffic with VXLAN, you can transport it over a Layer 3 IP Fabric, which has better scaling and high availability metrics than a traditional Layer 2 network. Some of the VXLAN features that Juniper QFX5100 switches supports are:

- OVSDB and VMware NSX control plane support
- DMI and Juniper Contrail control plane support
- VXLAN Layer 2 Gateway for bare-metal server support

Chapter 8 contains more in-depth content about VXLAN.

Ethernet

One of the most fundamental data center protocols is Ethernet. When a piece of data is transferred between end points, it's going to use Ethernet as the vehicle. The Juniper QFX5100 family supports all of the typical Ethernet protocols:

- IEEE 802.3
- IEEE 802.1Q
- IEEE 802.1QinQ

Pretty straightforward, eh?

FCoE

One of the biggest advantages of the Juniper QFX5100 series is the ability to support converged storage via FCoE. The two Juniper architectures that enable FCoE are QFabric and VCF. Figure 4-8 looks at how FCoE would work with VCF.

The servers would use standard Converged Network Adapters (CNA) and can be dual-homed into the VCF. Both data and storage would flow across these links using FCoE. The Storage Area Network (SAN) storage device would need to speak native Fibre Channel (FC) and use a pair of FC switches for redundancy. The FC switches would terminate into a pair of FC gateways that would convert FC into FCoE, and vice versa. In this scenario, VCF simply acts as a FCoE transit device. The FC gateway and switch functions need to be provided by other devices.

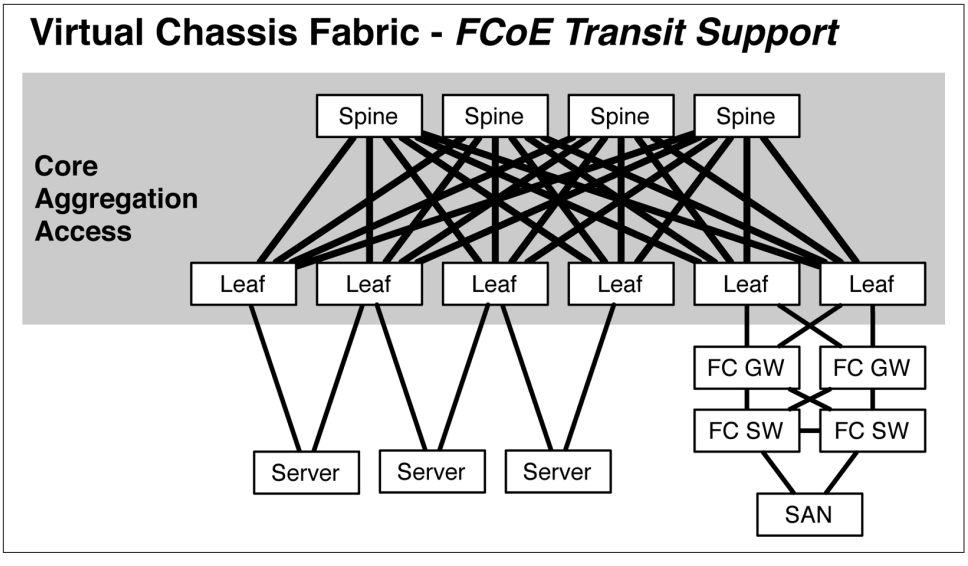


Figure 4-8. FCoE transit with VCF

HiGig2

One of the more interesting encapsulations is Broadcom HiGig2; this encapsulation can be used only between switches using a Broadcom chipset. The Broadcom HiGig2 is just another transport encapsulation, but the advantage is that it contains more fields and meta information that vendors can use to create custom architectures. For example, VCF uses the Broadcom HiGig2 encapsulation.

One of the distinct advantages of HiGig2 over standard Ethernet is that there's only a single ingress lookup. The architecture only needs to know the egress Broadcom chipset when transmitting data; any intermediate switches simply forward the HiGig2 frames to the egress chipset without having to waste time looking at other headers. Because the intermediate switches are so efficient, the end-to-end latency using HiGig2 is less than standard Ethernet.

The HiGig2 encapsulation isn't user-configurable. Instead this special Broadcom encapsulation is used in the following Juniper architectures: QFabric, Virtual Chassis, and VCF. This allows Juniper to offer options better performance and ease of use when building a data center. Juniper gives you the option to "do it yourself" with all the standard networking protocols, and the "plug and play" option for customers who want a simplified network operations.

Summary

This chapter covered the six different technology options of the Juniper QFX5100 series. There are three Juniper architecture options:

- Virtual Chassis
- QFabric
- VCF

There are also three open architecture options:

- Standalone
- MC-LAG
- Clos Fabric

In addition to the six architectures supported by the Juniper QFX5100, there are five major transport encapsulations, as well:

- MPLS
- VXLAN
- Ethernet
- FCoE
- HiGig2

The Juniper QFX5100 family of switches is a great platform on which to standardize because each offers so much in a small package. You can build efficient Ethernet fabrics with QFabric or VCF; large IP Fabrics using a Clos architecture; and small WAN deployments using MPLS. Using a single platform has both operational and capital benefits. Being able to use the same platform across various architectures creates a great use case for sparing. And, keeping a common set of power supplies, modules, and switches for failures lowers the cost of ownership.

CHAPTER 5

Virtual Chassis Fabric

A growing trend in the networking industry is to move away from traditional architectures such as Layer 2–only access switches or putting both Layer 2 and Layer 3 in the distribution layer. The next logical step for ubiquitous Layer 2 and Layer 3 access with ease of management is to create an Ethernet fabric.

Virtual Chassis Fabric (VCF) is a plug-and-play Ethernet fabric that offers a single point of management and many, many features. Think of a 3-stage Clos topology with the look and feel like a single logical switch; this is another good way to visualize VCF.

Overview

It's a common myth that a high-performance, feature-rich network is difficult to manage. This usually stems from the fact that there are many factors that the administrator must worry about:

- Performance
- Scale
- Latency
- High availability
- Routing protocols
- Equal cost multipathing
- Layer 2 and Layer 3 access
- Lossless Ethernet
- Software upgrades

- Management

Such a laundry list of tasks and responsibilities is often daunting to a small or medium-sized company with a minimal IT staff, and indeed, without assistance, would be difficult to administer. VCF was created specifically to solve this problem. It provides an architecture by which a single person can manage the entire network as if it were a single device, without sacrificing the performance, high availability, or other features.

It's easy to assume from the name that VCF has a lot of roots in the original Virtual Chassis technology; if you made such an assumption, you would be correct. VCF expands on the original Virtual Chassis technology and introduces new topologies, features, and performance.

Architecture

One of the most compelling benefits of VCF is the ability to create 3-stage Clos topologies (see Figure 5-1). VCF is the encapsulation of all of the switches in the 3-stage Clos topology.

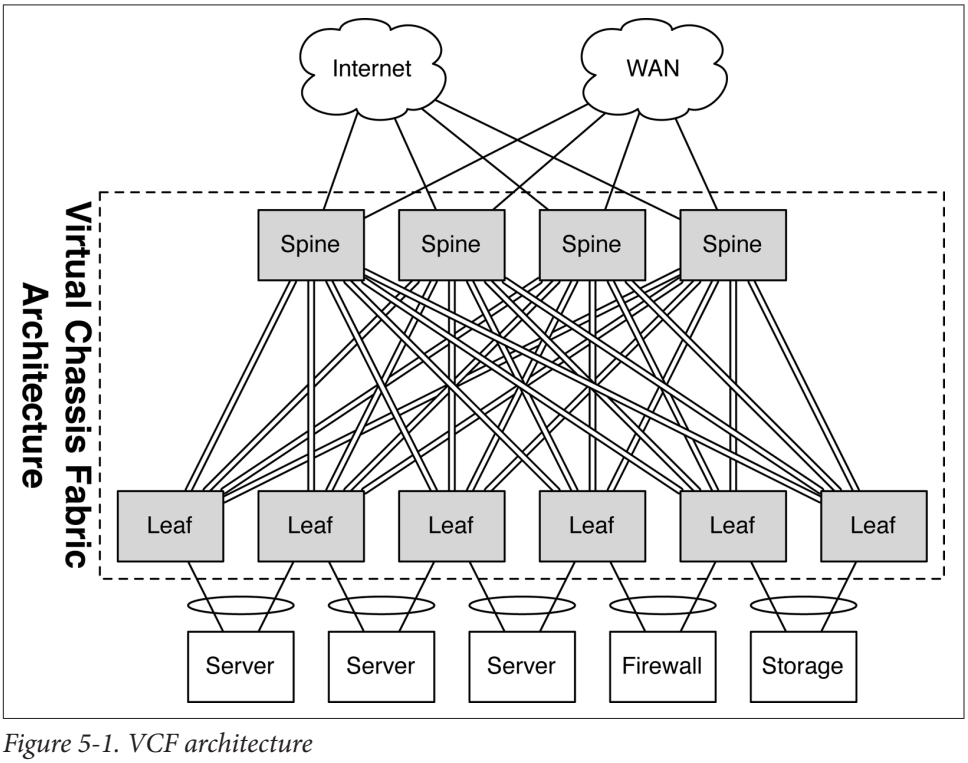


Figure 5-1. VCF architecture

There are basically two high-level roles in the VCF architecture: spine and leaf. The spine and leaf roles are used to create the 3-stage Clos topology and described here:

Spine
: The spine switches are at the heart of the topology and are used to interconnect all of the other leaf switches. Typically, the spine switches are higher-speed devices than leaf switches; this is to maintain low latency and high performance when looking at the entire network end-to-end.

Leaf
: The leaf switches are the ingress and egress nodes of the 3-stage Clos fabric. Most of the end points in the data center will connect through the leaf switches. The leaf switches are feature rich, can support servers, storage, and appliances, and can peer with other networking equipment.

The leaf switch role can support any of the Juniper QFX5100 switches to support various port densities and speeds; if you need a large deployment of 1GbE interfaces, you can use an EX4300 device as the leaf switch. VCF also offers investment protection; you can use existing switches such as those in the QFX3500 and QFX3600 families.

VCF is a flexible platform that allows you to incrementally change and increase the scale of the network. For example, you can start with two spine and two leaf switches today and then upgrade to four spine and 28 leaf switches tomorrow. Adding switches into the fabric is made very easy thanks to the plug-and-play nature of the architecture. You can add new leaf switches into the topology that are then automatically discovered and brought online.

Traffic engineering

VCF uses the Intermediate System to Intermediate System (IS-IS) routing protocol internally with some modified type length values (TLVs); this affords VCF a full end-to-end view of the topology, bandwidth, and network. As Juniper QFX5100 switches are combined together to create a VCF, the links connecting the switches automatically form logical links called *Smart Trunks*. As traffic flows across the VCF, the flows can be split up at each intersection in the fabric in an equal or unequal manner depending on the bandwidth of the links. For example, if all of the links in the VCF were the same speed and same quantity, all next-hops would be considered equal. However, during failure conditions, some links could fail and some switches could have more bandwidth than other switches. Smart Trunks allow for Unequal-Cost Multipathing (UCMP) in the event that some paths have more bandwidth than others. As a result, traffic is never dropped in a failure scenario.

Adaptive Load Balancing: Of Mice and... Elephants?

OK, it might not be Steinbeck-esque, but in the data center, there's a story about mice and elephants. The idea is that there are long-lived flows in the data center network

that consume a lot of bandwidth; these types of flows are referred to as *elephant flows*. Some examples of elephant flows are backups and copying data. Other types of flows are short-lived and consume little bandwidth; these are referred to as *mice flows*. Examples of mice flows include DNS and NTP.

The problem is that each flow within a network switch is subject to Equal-Cost Multi-path (ECMP), and is pinned to a particular next-hop or uplink interface. If a couple of elephant flows get pinned to the same interface and consume all of the bandwidth, they will begin to overrun other, smaller mice flows on the same egress uplink. Due to the nature of standard flow hashing, mice flows have a tendency to be trampled by elephant flows, which has a negative impact on application performance.

VCF has a very unique solution to the elephant and mice problem. If you take a closer look at TCP flows, you will notice something called *flowlets*. These are the blocks of data being transmitted between the TCP acknowledgement from the receiver. Depending on the bandwidth, TCP window size, and other variables, flowlets exist in different sizes and frequencies, as illustrated in Figure 5-2.

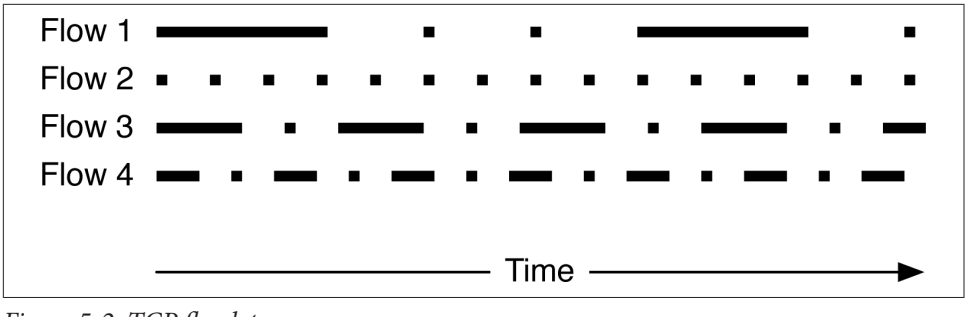


Figure 5-2. TCP flowlets

One method to solve the elephant-and-mice problem is to hash the flowlets to different next-hops. For example, in Figure 5-2, if Flow 1 and Flow 3 were elephant flows, each of the flowlets could be hashed to a different uplink, as opposed to the entire flow being stuck on a single uplink. VCF uses the flowlet hashing functionality to solve the elephant-and-mice problem; this feature is called Adaptive Load Balancing (ALB).

ALB is enabled by the use of a hash-bucket table (see Figure 5-3). The hash-bucket table has hundreds of thousands of entries; this is large enough to avoid any elephant flowlet collisions. As each flowlet egresses an interface, the hash-bucket table is updated with a timestamp and the egress link. For each packet processed, the time elapsed since the last packet received is compared with an inactivity timer threshold. If the last activity timer exceeds the threshold, the hash-bucket table and packet is assigned a new egress link. The eligibility of a new egress link indicates a new flowlet.

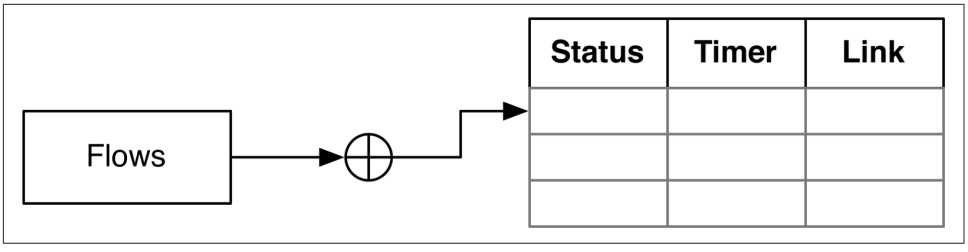


Figure 5-3. Flowlet hashing

Another important factor when selecting a new egress interface for a new flowlet is the link quality. The link quality is defined as a moving average of the link's load and queue depth. The link with the least utilization and congestion is selected as the egress interface for new flowlets.

ALB is disabled by default and must be turned on in the Junos configuration as shown below:

```
[edit]
root# set fabric-load-balance flowlet
root# commit
```

To ensure in-order packet delivery, the inactivity internal should be larger than the largest latency skew amount all the paths in the VCF from any node to any other node. The default inactivity timer is 16µs; the timer can be changed from 16µs to 32ms. The basis premise is that you do not want your inactivity-timer set too high, otherwise it won't be able to detect the flowlets. The best practice is to leave it set at the default value of 16µs.

To change the inactivity interval, use the following configuration:

```
[edit]
root# set fabric-load-balance flowlet inactivity-interval <value>
root# commit
```

The value can be specified in simple terms such as "20us" or "30ms." There's no need to convert the units into nanoseconds; just use the simple "us" and "ms" postfixes. To enable ALB to use any available next-hop based upon usage for ECMP, you may enable per-packet mode in ALB with the following configuration:

```
[edit]
root# set fabric-load-balance per-packet
root# commit
```

When per-packet mode is enabled, the VCF forwarding algorithm dynamically monitors all paths in VCF and forwards packets to destination switches using the best available path at that very moment. Flows are reordered when using per-packet mode, so some performance impact could be seen.

Requirements

There are a few key requirements that must be satisfied to create a VCF. Table 5-1 contains a listing of which switches can be used as a spine and leaf switch.

Table 5-1. VCF switch requirements

Switch	Spine	Leaf
QFX5100-24Q	Yes	Yes
QFX5100-96S	Yes	Yes
QFX5100-48S	Yes	Yes
QFX5100-48T	No	Yes
QFX3500	No	Yes
QFX3600	No	Yes
EX4300	No	Yes

VCF only supports 3-stage Clos topologies; other topologies might work but are not certified or supported by Juniper.

Software. Not all Junos software is compatible with VCF. You must use at least Junos 13.2X51-D20 or newer on all switches in the VCF.

Spine. A spine switch must be a QFX5100 switch; there are no exceptions. The amount of processing required in the spine requires additional control plane processing. The Juniper QFX5100 family has an updated control plane and makes a perfect fit for the spine in a VCF. Spine switches must also have a direct connection to each leaf switch in the topology. You cannot use intermediate switches or leave any leaf unconnected. The spine switches always assume the roles of the routing engine or the backup routing engine.

As of Junos 13.1X53-D10, VCF only supports up to four spine switches.

Leaf. Leaf switches are optimally Juniper QFX5100 switches, but there is also support for QFX3500, QFX3600, and EX4300 switches. Each leaf must have a direct connection to each spine in the topology. The leaf switches always assume the role of a line card.

As of Junos 14.1X53-D10, VCF supports up to 28 leaf switches.

Virtual Chassis modes

VCF supports two modes: fabric mode and mixed mode. By default, the switch ships in fabric mode. The mode is set on a per-switch basis. All switches in the VCF must be set to the same mode.

It's recommended that you change the mode of a switch before connecting it into a VCF. Connecting a new switch into the fabric and then changing the mode can cause temporary disruptions to the fabric.

Fabric mode. A VCF in fabric mode supports only QFX5100 devices. Fabric mode is the most recommended mode because it represents the latest technology, features, and scale. When the VCF is in fabric mode, it supports the full scale and features of the Juniper QFX5100 series.

Mixed mode. If you want to introduce native 1GbE connectivity with the EX4300 family or use existing QFX3500 and QFX3600 switches, the VCF must be placed into mixed mode. One of the drawbacks to using mixed mode is that VCF will operate in a "lowest common dominator" mode in terms of scale and features. For example, if you're using an EX4300 switch as a leaf in VCF, you would cause the entire fabric to operate at the reduced scale and feature level of the EX4300 device, as opposed to that of the Juniper QFX5100 device. The same is true for QFX3500 and QFX3600 switches.

Provisioning configurations

Provisioning a VCF involves how the fabric is configured on the command line as well as how new switches are added into the fabric. There are three modes associated with provisioning a VCF. Table 5-2 compares them at a high level.

Table 5-2. Comparing Virtual Chassis Fabric provisioning modes

Attribute	Auto-provisioned	Preprovisioned	Nonprovisioned
Configure serial number	Yes	Yes	Yes
Configure role	Yes	Yes	Yes
Configure priority	No	No	Yes
Adding new leaf	Plug-and-play	Configure role and serial number	Configure priority and serial number
Virtual Chassis ports	Automatic	Automatic	Manual

There are benefits and drawbacks to each provisioning mode. Use Table 5-2 and the sections that follow to understand each mode and make the best decision for your network. In general, it's recommended to use the auto-provisioned mode because it's a plug-and-play fabric.

Auto-provisioned mode. The easiest and recommended method is to use the auto-provisioned mode in VCF. There is minimal configuring required on the command line, and adding new switches into the fabric is as simple as connecting the cables and powering on the device; it's truly a plug-and-play architecture.

The only manual configuration required for auto-provisioned mode is to define the number of spine switches, set the role to `routing-engine`, and the serial number for each spine. For example:

```
[edit]
root# set virtual-chassis member 0 role routing-engine serial-number TB3714070330

[edit]
root# set virtual-chassis member 1 role routing-engine serial-number TB3714070064
```

In the preceding example, the VCF has two spine switches. We manually configured them as routing engines and set each serial number. Virtual Chassis ports are automatically discovered and configured in auto-provisioned mode.

Preprovisioned mode. The second most common method is the preprovisioned mode. The difference is that you must manually configure each switch in the topology, assign a role, and set the serial number. You cannot add new switches into the VCF without configuration. In environments with higher security requirements, a preprovisioned VCF would prevent unauthorized switches from being added into the fabric. The configuration is identical to the auto-provisioned mode. Virtual Chassis ports are automatically discovered and configured in preprovisioned mode.

Nonprovisioned mode. The nonprovisioned mode is the default configuration of each switch from the factory. The role is no longer required to be defined in this role; instead, a mastership election process is used to determine the role of each switch. The mastership election process is controlled through setting a priority on a per-switch basis. You define Virtual Chassis ports manually. They are not automatically discovered.

The nonprovisioned mode isn't recommended in general, and is only reserved for environments that require a specific mastership election process during a failure event. Adding new switches to the fabric requires serial number and priority configuration.

Components

At a high level, there are four components that are used by the switches to build a VCF: routing engine, line card, virtual management Ethernet interface, and Virtual Chassis ports.

Master routing engine

The first role a switch can have in a VCF is a routing engine. Only spine switches can become a routing engine. The leaf switches can only be a line card. The role of routing engine acts as the control plane for the entire VCF. A spine switch operating as a routing engine is responsible for the following:

- Operating as the control plane for the entire VCF
- Operating VCF control protocols for auto-discovery, topology management, and internal routing and forwarding protocols
- Taking ownership of the virtual management Ethernet interface for the VCF

Other spine switches must operate in the backup or line-card role. The next spine switch in succession to become the next master routing engine will operate in the backup role. All other spine switches will operate as line cards. In summary, only the first spine switch can operate as a master routing engine; the second spine switch operates as the backup routing engine; and the third and fourth spine switches operate as a line card.

Backup routing engine

The backup routing engine is similar to the master routing engine, except that its only job is to become the master routing engine if there's a failure with the current master routing engine. Part of this responsibility requires that the master and backup routing engines must be perfectly synchronized in terms of kernel and control plane state. The following protocols are used between the master and backup routing engines to keep synchronized:

- Graceful Routing Engine Switch Over (GRES)
- Nonstop Routing (NSR)
- Nonstop Bridging (NSB)

Keeping the backup routing engine synchronized with the master routing engine allows VCF to experience a hitless transition between the master and backup routing engines without traffic loss.

Line card

All other switches in the VCF that aren't a master or backup routing engine are a line card. By default, all leaf switches are a line card. If there are more than two spines, all other spines are also a line card; only the first two spines can be a routing engine.

The line card role acts simply as a line card would in a real chassis. There are minimal control plane functions on the line card to process the Virtual Chassis management and provisioning functions; otherwise, the switch simply exists to forward and route traffic as fast as possible.

Virtual Management Ethernet interface

Each switch in a VCF has a Management Ethernet (vme) port. These ports are used to manage the switch over IP. They are directly controlled by the routing engine and are

out-of-band from the revenue traffic. Figure 5-4 shows an example of the Virtual Management Ethernet interface, labeled C0 and C1.

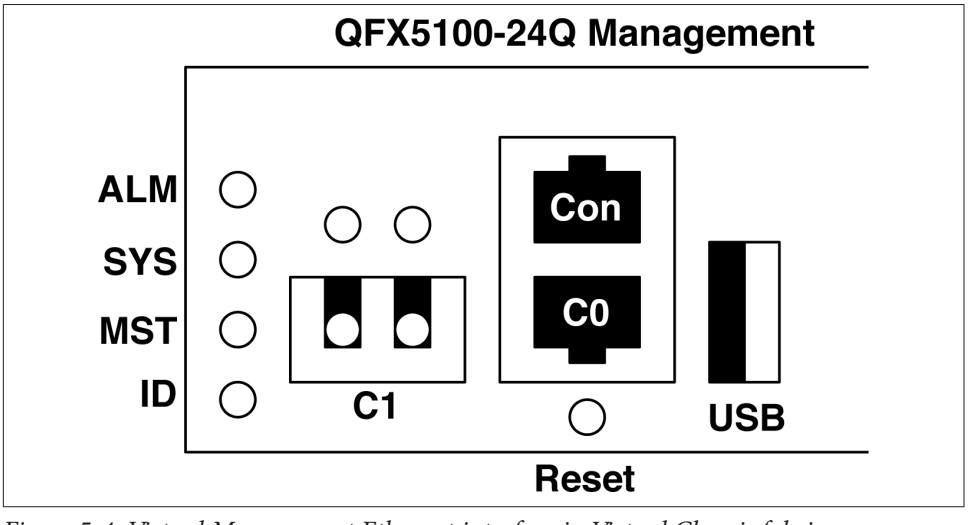


Figure 5-4. Virtual Management Ethernet interface in Virtual Chassis fabric

However, in a VCF, only one of these vme ports can be active at any given time. The switch that currently holds the master routing engine role is responsible for the vme management port. Although a VCF could have up to 32 switches, only a single switch will be used for out-of-band management through the vme port.

Virtual Chassis ports

The Virtual Chassis ports (VCP) are the interfaces that directly connect the switches together. VCP interfaces are standard 10GbE and 40GbE interfaces on the switch and do not require special cables. Simply use the existing QSFP and SFP+ interfaces to interconnect the switches together, as shown in Figure 5-5.

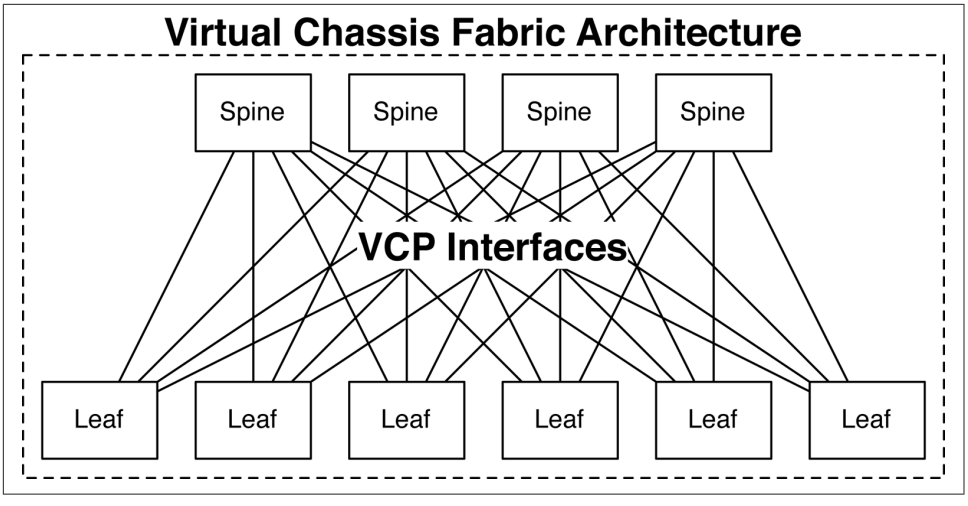


Figure 5-5. VCP interfaces in VCF

After an interface has been configured as a VCP interface, it's no longer eligible to be used as a standard revenue port. All interswitch traffic will now use VCP interfaces.

Implementation

Configuring VCF is straightforward and easy. Let's take a look at all three provisioning modes to get a better understanding of the configuration differences. We will also take a look at how to add and remove spine and leaf switches. Each provisioning mode is a little different in the configuration and process of expanding the fabric.

Before configuring the switches, there are a few steps that you are required to carry out before configuring the VCF.

Software Version
: Ensure that all switches have the same version of Junos installed. Use Junos 13.2X51D-20 or newer.

Disconnect All Cables
: Before you begin to configure VCF, be sure to disconnect all cables from the switches. This is because if you want to use the plug-and-play feature of the auto-provisioned mode, you want to explicitly control the creation of the spine switches, then simply add other switches. For preprovisioned and nonprovisioned modes, you can cable-up the switches.

Identify Serial Numbers
: Identify the serial numbers for each switch. For auto-provisioned mode, you only need the serial numbers for the spine switches. For preprovisioned and nonprovisioned modes, you will need all of the spine and leaf switch serial numbers.

Check for Link Layer Discovery Protocol (LLDP)
: LLDP should be turned on by factory default, but always check to ensure that it's enabled. Use the command **`set protocols lldp interface all`** to enable LLDP. VCF uses LLDP to enable the plug-and-play functionality in auto-provisioned mode.

After you have upgraded the software, disconnected all cables, and identified all of the serial numbers, you can begin to build the VCF.

Configuring the Virtual Management Ethernet interface

Now, let's configure a management IP address for this switch:

```
{master:0}[edit]
root# set interfaces vme.0 family inet address 10.92.82.4/24
```

The next step is to set a root password for the switch. It will prompt you to enter a password, and then again for verification:

```
{master:0}[edit]
root# set system root-authentication plain-text-password
New password:
Retype new password:
```

The next step is to enable Secure Shell (SSH) so that we can log in and copy files to and from the switch:

```
{master:0}[edit]
root# set system services ssh root-login allow
```

Now that you have set a management IP address and root password, you need to commit the changes to activate them:

```
{master:0}[edit]
root# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode

root>
```

The switch should now be reachable on the C0 management interface on the rear of the switch. Let's ping it to double-check:

```
epitaph:~ dhanks$ ping 10.92.82.4
PING 10.92.82.4 (10.92.82.4): 56 data bytes
64 bytes from 10.92.82.4: icmp_seq=0 ttl=55 time=21.695 ms
64 bytes from 10.92.82.4: icmp_seq=1 ttl=55 time=20.222 ms
^C
--- 10.92.82.4 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 20.222/20.959/21.695/0.737 ms
epitaph:~ dhanks$
```

Everything looks great. The switch is now upgraded and able to be managed via IP instead of the serial console.

Auto-provisioned

We'll spend more time going over the auto-provisioned mode in more detail because it's the most popular and recommended provisioning mode. The auto-provisioned mode only requires that you define the spine switches and the serial numbers. Figure 5-6 presents a simple topology and what the configuration would be.

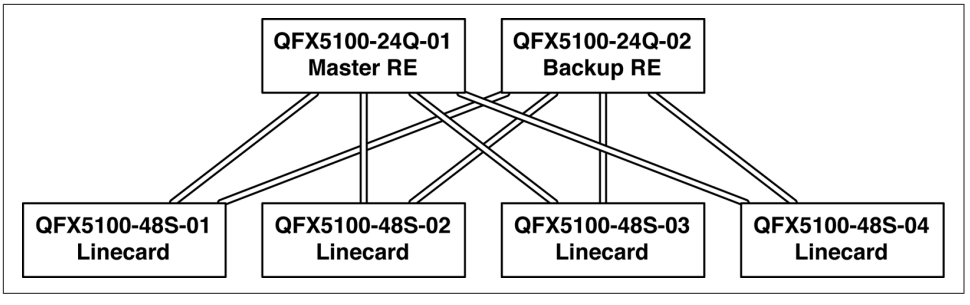


Figure 5-6. A simple VCF topology

The topology in Figure 5-6 has two spines and four leaf switches. In this example, both spine switches need to be configured. The spine switch serial numbers have been identified and are shown in Table 5-3.

Table 5-3. QFX5100-24Q spine serial numbers

Switch	Serial number
QFX5100-24Q-01	TB3714070330
QFX5100-24Q-02	TB3714070064

Installing the first spine switch. The first step is to ensure that the spine switches are in fabric mode. Use the following command on both QFX5100-24Q switches:

```
root> request virtual-chassis mode fabric reboot
```

The switches will reboot to fabric mode.

The next step is to begin configuring VCF on the first spine. Put QFX5100-24Q-01 into auto-provisioned mode. We'll also support upgrading the software of other switches connected into the fabric with the command `auto-sw-upgrade knob`.

Don't worry about the second spine switch, QFX5100-24Q-02, for the moment. We'll focus on QFX5100-24Q-01 and move on to the leaf switches. Adding the final spine switch will be the last step when bringing up VCF.

Starting on QFX5100-24Q-01, let's begin to configure VCF:

```
[edit]
root# set virtual-chassis auto-provisioned
[edit]
root# set virtual-chassis auto-sw-upgrade
```

The next step is to configure the role and serial numbers of all of the spine switches (use the data presented earlier in Table 5-3):

```
[edit]
root# set virtual-chassis member 0 role routing-engine serial-number TB3714070330

[edit]
root# set virtual-chassis member 1 role routing-engine serial-number TB3714070064
```

Verify the configuration before committing it:

```
[edit]
root# show virtual-chassis
auto-provisioned;
member 0 {
    role routing-engine;
    serial-number TB3714070330;
}
member 1 {
    role routing-engine;
    serial-number TB3714070064;
}
```

Now, you can commit the configuration:

```
[edit]
root# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode
```

The Juniper QFX5100-24Q is now in VCF mode; we can verify this by using the **`show virtual-chassis`** command:

```
{master:0}
root> show virtual-chassis

Fabric ID: 5ba4.174a.04ca
Fabric Mode: Enabled
                                                Mstr           Mixed Route Neighbor List
Member ID  Status   Serial No    Model          prio  Role      Mode  Mode ID  Interface
0 (FPC 0)  Prsnt    TB3714070330 qfx5100-24q-2p 128   Master*      N  F    0  vcp-255/0/0
```

Installing the first leaf switch. The next step is to begin installing the leaves, which is a very simple process. Log in to the first QFX5100-48S-01 and reset the switch to a factory default state:

```
root> request system zeroize
warning: System will be rebooted and may not boot without configuration
Erase all data, including configuration and log files? [yes,no] (no) yes
```

```
warning: ipsec-key-management subsystem not running - not needed by configuration.

root> Terminated
```

If the leaf switches already have Junos 13.2X51-D20 installed and are in a factory default state, you can skip the **`request system zeroize`** step. You can simply connect the leaf switch to the spine switch.

After the switch reboots, simply connect a 40G cable from the Juniper QFX5100-24Q-01 to QFX5100-48S-01, as illustrated in Figure 5-7.

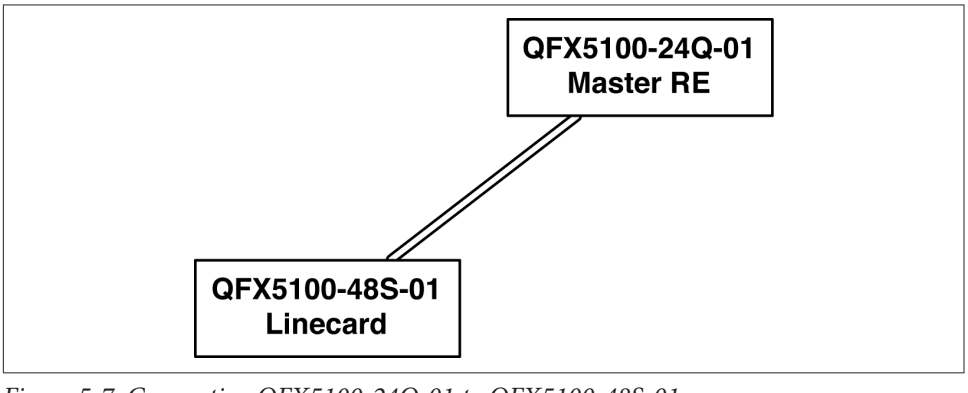


Figure 5-7. Connecting QFX5100-24Q-01 to QFX5100-48S-01

When the cable is connected, the master QFX5100-24Q-01 will automatically add the new QFX5100-48S-01 into the VCF.

Install remaining leaf switches. Repeat this step for each QFX5100-48S in the VCF, as shown in Figure 5-8.

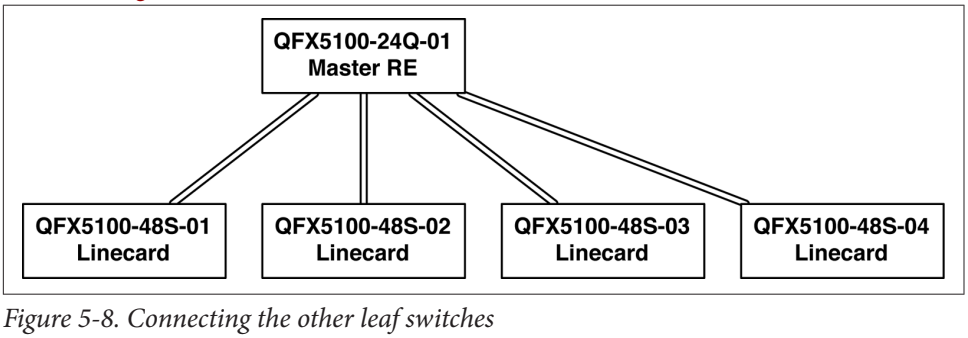


Figure 5-8. Connecting the other leaf switches

When all of the Juniper QFX5100-48S leaves have been reset to factory default and connected, the Juniper QFX5100-24Q-01 will bring all of the switches into the VCF. You can verify this by using the **`show virtual-chassis`** command:

```
{master:0}
root> show virtual-chassis

Auto-provisioned Virtual Chassis Fabric
Fabric ID: 742a.6f8b.6de6
Fabric Mode: Enabled
                                                Mstr           Mixed Route Neighbor List
Member ID  Status   Serial No    Model          prio  Role      Mode  Mode ID  Interface
0 (FPC 0)  Prsnt    TB3714070330 qfx5100-24q-2p 129   Master*      N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
                                                                          2  vcp-255/0/3
                                                                          5  vcp-255/0/4
2 (FPC 2)  Prsnt    TA3713480228 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
3 (FPC 3)  Prsnt    TA3713480106 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
4 (FPC 4)  Prsnt    TA3713470455 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
5 (FPC 5)  Prsnt    TA3713480037 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
```

The newly added switches are displayed in italics in the preceding output; for reference they're Member ID 2, 3, 4, and 5.

Install the last spine. The last step is to add the second QFX5100-24Q-02 spine into the VCF. Repeat the same steps and reset the switch using the `zeroize` command on the second QFX5100-24Q-02, and then after the switch reboots, connect the remaining cables into a full mesh, as depicted in Figure 5-9.

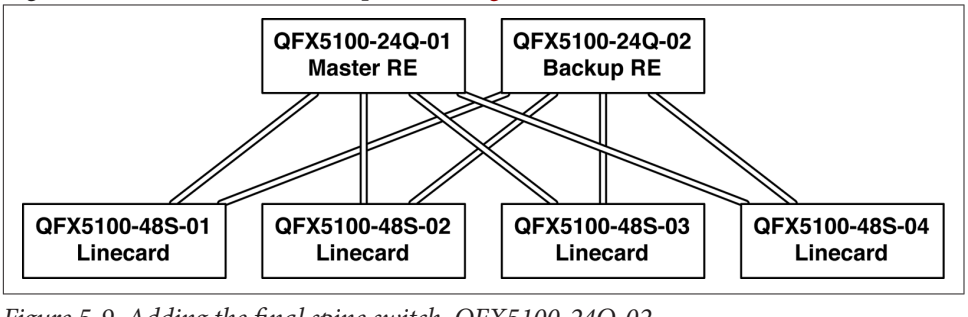


Figure 5-9. Adding the final spine switch, QFX5100-24Q-02

Wait a couple of minutes and then check the status of the VCF again; you should see the missing member 1 as `Prsnt` with a role of `Backup`:

```
dhanks@> show virtual-chassis

Auto-provisioned Virtual Chassis Fabric
Fabric ID: 742a.6f8b.6de6
Fabric Mode: Enabled
                                                Mstr           Mixed Route Neighbor List
Member ID  Status   Serial No    Model          prio  Role      Mode  Mode ID  Interface
0 (FPC 0)  Prsnt    TB3714070330 qfx5100-24q-2p 129   Master*      N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
```

```
                                                                                2  vcp-255/0/3
                                                                                5  vcp-255/0/4
1 (FPC 1)  Prsnt    TB3714070064 qfx5100-24q-2p 129   Backup       N  F    4  vcp-255/0/0
                                                                                3  vcp-255/0/1
                                                                                2  vcp-255/0/3
                                                                                5  vcp-255/0/4
2 (FPC 2)  Prsnt    TA3713480228 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                                1  vcp-255/0/49
3 (FPC 3)  Prsnt    TA3713480106 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                                1  vcp-255/0/49
4 (FPC 4)  Prsnt    TA3713470455 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                                1  vcp-255/0/49
5 (FPC 5)  Prsnt    TA3713480037 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                                1  vcp-255/0/49
```

Use the **`show interface`** command to verify that the new Virtual Chassis Fabric management interface is up:

```
{master:0}
root> show interfaces terse vme
Interface               Admin Link Proto    Local                 Remote
vme                     up    up
vme.0                   up    up   inet     10.92.82.4/24
```

You probably recognized (astutely, I should mention) that this is the same vme interface that we originally configured on the Juniper QFX5100-24Q-01 when it was in standalone mode. The vme configuration has persisted through when placing the device into VCF. Because the Juniper QFX5100-24Q-01 is the master routing engine, it also owns the vme interface. We can also check the reachability from our laptop:

```
epitaph:~ dhanks$ ping 10.92.82.4
PING 10.92.82.4 (10.92.82.4): 56 data bytes
64 bytes from 10.92.82.4: icmp_seq=0 ttl=55 time=21.695 ms
64 bytes from 10.92.82.4: icmp_seq=1 ttl=55 time=20.222 ms
^C
--- 10.92.82.4 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 20.222/20.959/21.695/0.737 ms
epitaph:~ dhanks$
```

It appears that we can reach the VCF by using the built-in management port. We're now ready for the next step.

Configure high availability. To ensure that the VCF recovers quickly from failures, there are three key features that we need to enable:

- **GRES:** Synchronize kernel state between the master and backup routing engines
- **NSR:** Synchronize routing protocol state between the master and backup routing engines
- **NSB:** Synchronize Layer 2 protocol state between the master and backup routing engines

The first step is to configure GRES:

```
{master:0}[edit]
dhanks@VCF# set chassis redundancy graceful-switchover
{master:0}[edit]
dhanks@VCF# set system commit synchronize
```

Next, configure NSR and NSB:

```
{master:0}[edit]
dhanks@VCF# set routing-options nonstop-routing
{master:0}[edit]
dhanks@VCF# set protocols layer2-control nonstop-bridging
{master:0}[edit]
dhanks@VCF# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode
```

Now, verify that the master routing engine is sending data to the backup routing engine through the GRES protocol:

```
{master:0}
dhanks@VCF> show task replication
        Stateful Replication: Enabled
        RE mode: Master
```

The next step is to verify that NSR and NSB are synchronizing state. To do this, you need to log in to the backup routing engine by using the **`request session`** command:

```
{master:0}
dhanks@VCF> request session member 1

--- JUNOS 13.2-X51D20
dhanks@VCF:BK:1% clear
dhanks@VCF:BK:1% cli
warning: This chassis is operating in a non-master role as part of a virtual-
chassis fabric (VCF) 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 Fabric Master (VCF-M)
chassis.
warning: The VCF-M can be identified through the show fabric status command
executed at this console.
warning: Please logout and log into the VCF-M to use CLI.
```

Now that you've logged in to the backup routing engine, verify NSR and NSB:

```
{backup:1}
dhanks@VCF> show system switchover
fpc1:
--------------------------------------------------------------------------
Graceful switchover: On
Configuration database: Ready
Kernel database: Ready
Peer state: Steady State

{backup:1}
dhanks@VCF> show l2cpd task replication
        Stateful Replication: Enabled
        RE mode: Backup
```

Everything looks great. At this point, VCF is configured and ready to use.

Preprovisioned

Configuring the preprovisioned VCF is very similar to the auto-provisioned method. Begin by configuring the following items just as you would for auto-provisioned mode:

- Ensure that switches are running Junos 13.2X51-D20 or higher
- Identify all of the serial numbers for both spine and leaf switches
- Disconnect all cables
- Configure the vme interface on the first spine switch and check connectivity

The next step is to begin configuring VCF in preprovisioned mode.

Starting on QFX5100-24Q-01, begin to configure VCF:

```
[edit]
root# set virtual-chassis preprovisioned
[edit]
root# set virtual-chassis auto-sw-upgrade
```

Configure the role and serial numbers of all of the spine switches (use the data provided in Table 5-3):

```
[edit]
root# set virtual-chassis member 0 role routing-engine serial-number TB3714070330

[edit]
root# set virtual-chassis member 1 role routing-engine serial-number TB3714070064
```

Configure the role and serial numbers of all of the leaf switches from Figure 5-6:

```
[edit]
root# set virtual-chassis member 2 role routing-engine serial-number TB3714070228

[edit]
root# set virtual-chassis member 3 role routing-engine serial-number TB3714070106

[edit]
root# set virtual-chassis member 4 role routing-engine serial-number TB3714070455

[edit]
root# set virtual-chassis member 5 role routing-engine serial-number TB3714070037
```

The next step is to connect the rest of the switches in the topology and turn them on.

Wait a couple of minutes and check the status of the VCF again; you should see the Virtual Chassis up and running:

```
dhanks@> show virtual-chassis

Pre-provisioned Virtual Chassis Fabric
Fabric ID: 742a.6f8b.6de6
```

```
Fabric Mode: Enabled
                                                Mstr           Mixed Route Neighbor List
Member ID  Status   Serial No    Model          prio  Role      Mode  Mode ID  Interface
0 (FPC 0)  Prsnt    TB3714070330 qfx5100-24q-2p 129   Master*     N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
                                                                          2  vcp-255/0/3
                                                                          5  vcp-255/0/4
1 (FPC 1)  Prsnt    TB3714070064 qfx5100-24q-2p 129   Backup      N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
                                                                          2  vcp-255/0/3
                                                                          5  vcp-255/0/4
2 (FPC 2)  Prsnt    TA3713480228 qfx5100-48s-6q   0   Linecard    N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
3 (FPC 3)  Prsnt    TA3713480106 qfx5100-48s-6q   0   Linecard    N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
4 (FPC 4)  Prsnt    TA3713470455 qfx5100-48s-6q   0   Linecard    N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
5 (FPC 5)  Prsnt    TA3713480037 qfx5100-48s-6q   0   Linecard    N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
```

From this point forward configure the high availability just as you did in the auto-provisioned mode. Ensure that the high-availability protocols are working and the routing engines are synchronized. From this point forward, you're good to go.

Nonprovisioned

Configuring the nonprovisioned VCF is very similar to the preprovisioned method. You begin by configuring the following items just as you would for preprovisioned mode:

- Ensure that switches are running Junos 13.2X51-D20 or higher
- Disconnect all cables
- Identify all serial numbers
- Configure the vme interface on the first spine switch and check connectivity

The next step is to begin configuring VCF in nonprovisioned mode.

Starting on QFX5100-24Q-01, begin to configure VCF:

```
 [edit]
root# set virtual-chassis auto-sw-upgrade
```

Now, configure the mastership priority, role, and serial numbers of all of the spine switches (use the data provided in Table 5-3):

```
[edit]
root# set virtual-chassis member 0 role routing-engine serial-number TB3714070330

[edit]
root# set virtual-chassis member 0 mastership-priority 255

[edit]
root# set virtual-chassis member 1 role routing-engine serial-number TB3714070064
```

```
[edit]
root# set virtual-chassis member 1 mastership-priority 254
```

The next step is to configure the mastership priority, role, and serial numbers of all of the leaf switches from Figure 5-6:

```
[edit]
root# set virtual-chassis member 2 role routing-engine serial-number TB3714070228

[edit]
root# set virtual-chassis member 2 mastership-priority 0

[edit]
root# set virtual-chassis member 3 role routing-engine serial-number TB3714070106

[edit]
root# set virtual-chassis member 3 mastership-priority 0

[edit]
root# set virtual-chassis member 4 role routing-engine serial-number TB3714070455

[edit]
root# set virtual-chassis member 4 mastership-priority 0

[edit]
root# set virtual-chassis member 5 role routing-engine serial-number TB3714070037

[edit]
root# set virtual-chassis member 5 mastership-priority 0
```

The mastership priority ranges from 0 to 255. The higher the mastership priority, the more priority it has to become the master routing engine.

Now, connect the rest of the switches in the topology and turn them on.

Wait a couple of minutes and check the status of the VCF again; you should see the Virtual Chassis up and running:

```
dhanks@> show virtual-chassis

Pre-provisioned Virtual Chassis Fabric
Fabric ID: 742a.6f8b.6de6
Fabric Mode: Enabled
                                                Mstr           Mixed Route Neighbor List
Member ID  Status   Serial No    Model          prio  Role      Mode  Mode ID  Interface
0 (FPC 0)  Prsnt    TB3714070330 qfx5100-24q-2p 129   Master*      N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
                                                                          2  vcp-255/0/3
                                                                          5  vcp-255/0/4
1 (FPC 1)  Prsnt    TB3714070064 qfx5100-24q-2p 129   Backup       N  F    4  vcp-255/0/0
                                                                          3  vcp-255/0/1
                                                                          2  vcp-255/0/3
                                                                          5  vcp-255/0/4
2 (FPC 2)  Prsnt    TA3713480228 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
3 (FPC 3)  Prsnt    TA3713480106 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
4 (FPC 4)  Prsnt    TA3713470455 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                          1  vcp-255/0/49
```

```
5 (FPC 5)  Prsnt    TA3713480037 qfx5100-48s-6q   0   Linecard     N  F    0  vcp-255/0/48
                                                                         1  vcp-255/0/49
```

From this point forward, configure the high availability just as we did in the auto-provisioned mode. Ensure that the high-availability protocols are working and the routing engines are synchronized. From this point forward, you're good to go.

Using Virtual Chassis Fabric

Now that VCF is configured and ready to go, let's take a look at some of the most common day-to-day tasks and how they work in VCF.

- Adding new Virtual Local Area Networks (VLANs) and assigning them to switch ports
- Assigning routed VLAN interfaces so that the fabric can route between VLANs
- Adding access control lists
- Mirroring traffic
- Setting up Simple Network Management Protocol (SNMP) to enable monitoring of the fabric

Remember that VCF is just a single, logical switch with many physical components. You handle all configuration through a single command-line interface. The VCF also appears as a single, large switch from the perspective of SNMP.

We'll make the assumption that our VCF has the following topology, as shown in Figure 5-10.

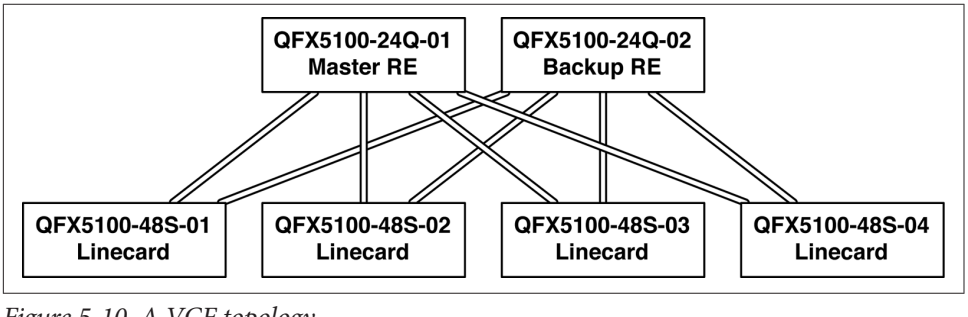


Figure 5-10. A VCF topology

Adding VLANs

The most basic task is adding new VLANs to the network in order to segment servers. The first step is to drop into configuration mode and define the VLAN:

```
{master:0}[edit]
root@VCF# set vlans Engineering description "Broadcast domain for Engineering
```

```
group"
{master:0}[edit]
root@VCF# set vlans Engineering vlan-id 100
{master:0}[edit]
root@VCF# set vlans Engineering l3-interface irb.100
```

The next step is to create a Layer 3 interface for the new Engineering VLAN so that servers have a default gateway. We'll use the same `l3-interface` that was referenced in creating the Engineering VLAN:

```
{master:0}[edit]
root@VCF# set interfaces irb.100 family inet address 192.168.100.1/24
```

Now that the VLAN and its associated Layer 3 interface is ready to go, the next step is to add servers into the VLAN. Let's make an assumption that the first QFX5100-48S is in the first rack.

When working with VCF, each switch is identified by its FPC number. An easy way to reveal a switch's FPC number is by using the **`show chassis hardware`** command. You can identify switches by the serial number. It's important to note that because we used the auto-provision feature in VCF, the FPC numbers are assigned chronologically as new switches are added:

```
{master:0}
root@VCF> show chassis hardware | match FPC
FPC 0            REV 11   650-049942   TB3714070330      QFX5100-24Q-2P
FPC 1            REV 11   650-049942   TB3714070064      QFX5100-24Q-2P
FPC 2            REV 09   650-049937   TA3713480228      QFX5100-48S-6Q
FPC 3            REV 09   650-049937   TA3713480106      QFX5100-48S-6Q
FPC 4            REV 09   650-049937   TA3713470455      QFX5100-48S-6Q
FPC 5            REV 09   650-049937   TA3713480037      QFX5100-48S-6Q
```

In our example, the FPC numbers are sequential, starting from 0 and ending in 5, as shown in Figure 5-11.

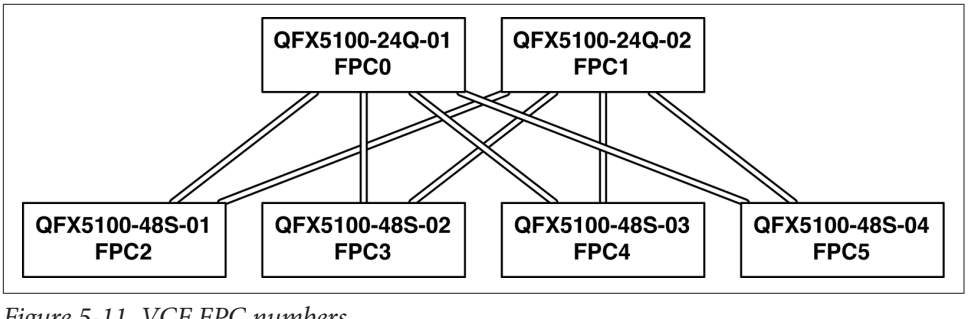


Figure 5-11. VCF FPC numbers

Now that we know that the first switch is FPC2, we can begin to assign the new Engineering VLAN to this switch. The easiest method is to create an alias for all of the 10GbE interfaces on this switch; we'll call this alias rack-01:

```
{master:0}[edit]
root@VCF# set interfaces interface-range rack-01 member-range xe-2/0/0 to xe-2/0/47
{master:0}[edit]
root@VCF# set interfaces interface-range rack-01 description "Alias for all 10GE
interfaces on FPC2/rack-02"
{master:0}[edit]
root@VCF# set interfaces interface-range rack-01 unit 0 family ethernet-switching
vlan members Engineering
{master:0}[edit]
root@VCF# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode
```

Now the new interface alias called rack-01 is configured to include all 10GbE interfaces from `xe-0/0/0` to `xe-0/0/47` on the front panel. The next step is to assign the Engineering VLAN, which is done via the **`vlan members`** command.

Let's verify our work by using the **`show vlans`** command:

```
{master:0}
root@VCF> show vlans

Routing instance        VLAN name             Tag          Interfaces
default-switch          Engineering           100
                                                             xe-2/0/0.0
                                                             xe-2/0/1.0
                                                             xe-2/0/12.0
                                                             xe-2/0/13.0
                                                             xe-2/0/2.0
                                                             xe-2/0/3.0
                                                             xe-2/0/4.0
                                                             xe-2/0/5.0
                                                             xe-2/0/6.0
                                                             xe-2/0/7.0
default-switch          default               1
```

All of the interfaces that have optics in rack-01 are now assigned to the Engineering VLAN.

Let's add another VLAN on a different switch for System Test:

```
{master:0}[edit]
root@VCF# set vlans Systest description "Broadcast domain for System Test"
{master:0}[edit]
root@VCF# set vlans Systest vlan-id 200
{master:0}[edit]
root@VCF# set vlans Systest l3-interface irb.200
{master:0}[edit]
root@VCF# set interfaces irb.200 family inet address 192.168.200.1/24
{master:0}[edit]
root@VCF# set interfaces interface-range rack-02 member-range xe-3/0/0 to xe-3/0/47
{master:0}[edit]
root@VCF# set interfaces interface-range rack-02 description "Alias for all 10GE
interfaces on FPC3/rack-03"
{master:0}[edit]
root@VCF# set interfaces interface-range rack-02 unit 0 family ethernet-switching
vlan members Systest
```

```
{master:0}[edit]
root@VCF# commit and-quit
configuration check succeeds
commit complete
Exiting configuration mode
```

We can verify that the new System Test VLAN is up and working with a couple of show commands:

```
{master:0}
root@VCF> show vlans

Routing instance        VLAN name             Tag          Interfaces
default-switch          Engineering           100
                                                             xe-2/0/0.0
                                                             xe-2/0/1.0
                                                             xe-2/0/12.0
                                                             xe-2/0/13.0
                                                             xe-2/0/2.0
                                                             xe-2/0/3.0
                                                             xe-2/0/4.0
                                                             xe-2/0/5.0
                                                             xe-2/0/6.0
                                                             xe-2/0/7.0
default-switch          Systest               200
                                                             xe-3/0/0.0
                                                             xe-3/0/1.0
                                                             xe-3/0/12.0
                                                             xe-3/0/13.0
                                                             xe-3/0/2.0
                                                             xe-3/0/3.0
                                                             xe-3/0/4.0
                                                             xe-3/0/5.0
                                                             xe-3/0/6.0
                                                             xe-3/0/7.0
default-switch          default               1

{master:0}
root@VCF> show interfaces terse | match irb
irb                     up    up
irb.100                 up    down inet     192.168.100.1/24
irb.200                 up    down inet     192.168.200.1/24
```

Configuring SNMP

With the VCF configured and running, the next step is to integrate the fabric into a network monitoring program. One of the most common ways to poll information from a switch is using SNMP. Let's set up a public community string with read-only access:

```
{master:0}[edit]
root@VCF# set snmp community public authorization read-only

{master:0}[edit]
root@VCF# commit and-quit
configuration check succeeds
commit complete
```

At this point, you can use your favorite SNMP browser or collection server and begin polling information from VCF. To confirm that SNMP is working properly, you can use the command-line tool `snmpwalk` and use the `vme0` management IP address and the `public` community string.

```
epitaph:~ dhanks$ snmpwalk -c public 10.92.82.4 | grep SNMPv2-MIB
SNMPv2-MIB::sysDescr.0 = STRING: Juniper Networks, Inc. qfx5100-24q-2p Ethernet
Switch, kernel JUNOS 13.2-X51-D20, Build date: 2014-03-18 12:13:29 UTC Copyright
(c) 1996-2014 Juniper Networks, Inc.
...
```

Port Mirroring

There are various ways to mirror traffic within VCF. You define an input and an output interface. The input is a bit more flexible and supports an interface or an entire VLAN. Let's set up a port mirror so that all ingress traffic on the Engineering VLAN is mirrored to the `xe-3/0/0.0` interface:

```
{master:0}[edit]
root@VCF# set forwarding-options analyzer COPY-ENGINEERING input ingress vlan
Engineering
root@VCF# set forwarding-options analyzer COPY-ENGINEERING output interface xe-
3/0/0.0
root@VCF# commit and-quit
configuration check succeeds
commit complete
```

To view and verify the creation of the analyzer, we can use the **`show forwarding-options analyzer`** command:

```
{master:0}
root@VCF> show forwarding-options analyzer
  Analyzer name                   : COPY-ENGINEERING
  Mirror rate                     : 1
  Maximum packet length           : 0
  State                           : up
  Ingress monitored VLANs         : default-switch/Engineering
```

Summary

VCF is a great technology to quickly get you on your feet and build out a high-performance network that you can managed as a single switch. VCF offers three provisioning modes to suit your data center management and security needs. Taking advantage of the carrier-class Junos code to provide GRES, NSR, and NSB, Virtual Chassis Fabric can gracefully switch between routing engines during a failure without dropping your critical traffic in the data center.

Whether you're building out a small to medium data center or a large data center with a POD architecture, VCF is a great way to easily manage your data center with a rich set of features and outstanding performance.

Chapter Review Questions

1. How many provisioning modes does VCF support?

a. 1

b. 2

c. 3

d. 4

2. How many vme interfaces are active at any given time in VCF?

a. None.

b. Only one.

c. Two. One for the master routing engine and another for the backup routing engine.

d. All of them.

3. Can you add an EX4300 switch to VCF in fabric mode?

a. Yes

b. No

4. You want to configure the first 10GbE port on the second leaf. What interface is this?

a. xe-1/1/0

b. xe-0/1/0

c. xe-1/1/1

d. xe-3/0/0

Chapter Review Answers

*1. **Answer: C.** VCF supports three provisioning modes: auto-provisioned, preprovisioned, and nonprovisioned.*

*2. **Answer: B.** Only the master routing engine's vme interface is active at any given time.*

*3. **Answer: B.** To support the EX4300, QFX3500, or the QFX3600, VCF must be put into mixed mode.*

*4. **Answer: D.** This is a really tricky question. It depends on how many spines there are. Using the assumption that there are two spines and four leaf switches, the FPC number of the second leaf switch would be 3. The first port would be 0. The answer would be xe-3/0/0.*

CHAPTER 6
Network Automation

Automating a network means many things to many people. It's like ordering a pizza, no one can agree on anything. That's because network automation is so personalized and specific to the problem that each person is solving, everyone's answer is different and focused on solving their own problem.

For the scope of this chapter, network automation will focus on the task of automating network functions. For simplicity, you can break down network functions into three simple categories:

Build
: The build stage focuses on the initial installation and bootstrapping of the networking equipment. As soon as the switch is racked and powered on, the build stage begins.

Configure
: After you've built the network, there are day-to-day changes that you need to implement to enable new services and applications to run; this is the configuration phase of network automation.

Collect
: Now that you have successfully built the network and have a good handle on the day-to-day operations, the last phase is to collect information about the network. Being able to understand what's happening on the network makes it possible for you to increase the availability of the network and quickly troubleshoot problems.

You can think of network automation as the purpose-built glue that brings your data center together. Some network automation tools offer more turnkey functionality than others; this is simply because of the scope. For example, to automatically configure a networking switch when it first powers on is a relatively simple thing to do;

such automation can be very turnkey. However, there are other examples such as deploying specific switch settings (Virtual Local Area Networks (VLAN) membership, interface tagging, and routing protocol changes in response to outside events including application provisioning or updates). Such specific examples require purpose-built programming that uses the switch's libraries and APIs in order to get the desired result—providing turnkey functionality is nearly impossible for such customized requirements.

This chapter will briefly go through the major automation tools that the Juniper QFX5100 supports. The problem with network automation is that an entire book can be written on each subject; so I've elected to give you a walking tour through the automation abilities of the Juniper QFX5100 series and references to where you can find additional information.

Overview

The Juniper QFX5100 family is chock full of network automation features with which you can carry out network-related activities faster. This chapter covers the key network automation tools that come standard on Juniper QFX5100 switches and describes how to use each tool. Following is an quick introduction:

Zero Touch Provisioning
: The first tool we cover is Zero Touch Provisioning; this tool makes it possible for you to bootstrap your switch when it first powers on and get it up and running automatically.

Chef and Puppet
: One of the most common tasks in a data center is making changes. Chef and Puppet are tools that enable engineers to provision changes across the entire data center, including the networking equipment.

Network Configuration Protocol and Device Management Interface
: The Network Configuration Protocol (NETCONF) is an IETF standard that's based on XML by which you can edit a network configuration over remote procedure calls (RPC). The Device Management Interface (DMI) is a Juniper-specific schema that defines all of the RPCs available within Junos that you can use.

Junos Python Easy Library
: The Junos Python Easy (PyEZ) library is a Python library that uses NETCONF and the Juniper DMI to automate Juniper devices. The best part is that the PyEZ library hides all of the NETCONF and DMI from the programmer and simply exposes standard Python functions as a replacement.

The Juniper QFX5100 series has many points of user interaction that are available for network automation, as shown in Figure 6-1. Chapter 1 describes how even the Junos CLI uses standard XML RPC commands in the background and displays formatted ASCII in the terminal. All of the other network automation points in Figure 6-1 are designed to be used by programming languages or automation tools.

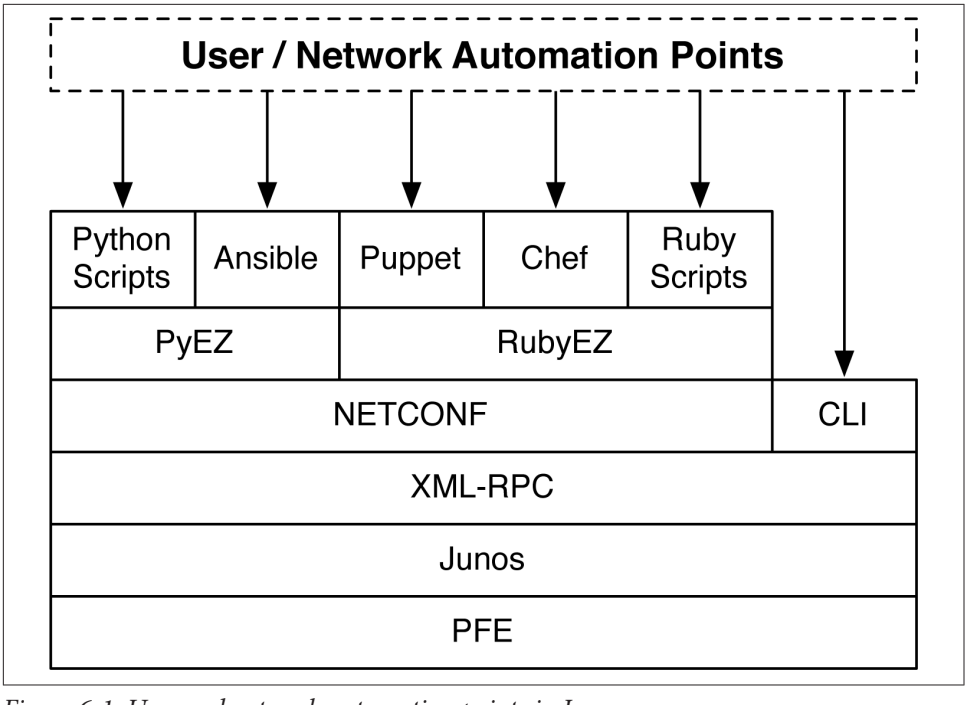


Figure 6-1. User and network automation points in Junos

All of the major network automation tools utilize the NETCONF protocol. However Junos isn't limited to the tools shown in Figure 6-1, and you can directly program the NETCONF protocol itself by using any of the following programming languages and libraries:

- Python (ncclient)
- Perl (netconf-perl)
- Ruby (net-netconf)
- Go (go-netconf)

Armed with your favorite programming language, there's no task too large or too small when it comes to network automation. The Juniper QFX5100 family supports a wide variety of programming languages and lends itself nicely to the DevOps com-

munity. The best way to learn is by doing. Later in the chapter, we'll get our hands dirty with the Junos PyEZ library.

Junos Enhanced Automation

As network automation is becoming more prevalent in the data center, Juniper Networks has created a new software package for the Juniper QFX5100 series of switches that comes preinstalled, offering additional programming and automation tools. The Junos Enhanced Automation software packages use the prefix "flex" to distinguish them from the standard Junos software packages. For example, *jinstall-qfx-5-flex-13.2X51-D20.2-domestic-signed.tgz* denotes Junos Enhanced Automation. The following changes have been made to Junos Enhanced Automation:

- It now maintains full-feature parity with the standard version of Junos.
- The factory default configuration is Layer 3 instead of Layer 2.
- Safeguards are in place to prevent changes to essential Junos files.
- ZTP is preconfigured for all management and server ports.
- A new 1 GB */user* partition is available to store binaries and additional packages.
- The */user* partition is never modified during an upgrade or downgrade.
- Python is preinstalled into */usr/bin/python*.
- Ruby is preinstalled into */opt/sdk/juniper/bin/ruby*.
- The Puppet agent is preinstalled into */opt/sdk/juniper/bin/puppet*.
- The Chef agent is preinstalled into */opt/sdk/chef/bin/chef*.

Network automation with Juniper QFX5100 devices is now as easy as pie when paired with Junos Enhanced Automation. Everything ships preinstalled so you don't have to worry about signed binaries from Juniper Networks and other headaches. As you begin building up a personal programming library and script repository, keep them installed in the */user* partition. As you upgrade the switch in the future, this guarantees that all of your files are never deleted.

Zero Touch Provisioning

So, you just received a large pallet of Juniper QFX5100 switches in the data center shipping department. You've removed all of the equipment from their boxes and installed them into the racks. Now what? Most network engineers power up everything and begin programming each switch by hand. The more experienced engineer is too savvy for this; he simply configures the first switch and saves the configuration to a text file. Every other switch in the network is configured by using this as a template, copying and pasting it into the RS-232 terminal.

If you do this, stop. There are much better ways to bootstrap networking equipment in the data center: Zero Touch Provisioning (ZTP)

Seriously, it does exactly what it says. ZTP configures the switch without you having to touch it. You simply power up the switch, connect its cable, and it will automatically configure itself as Figure 6-2 demonstrates.

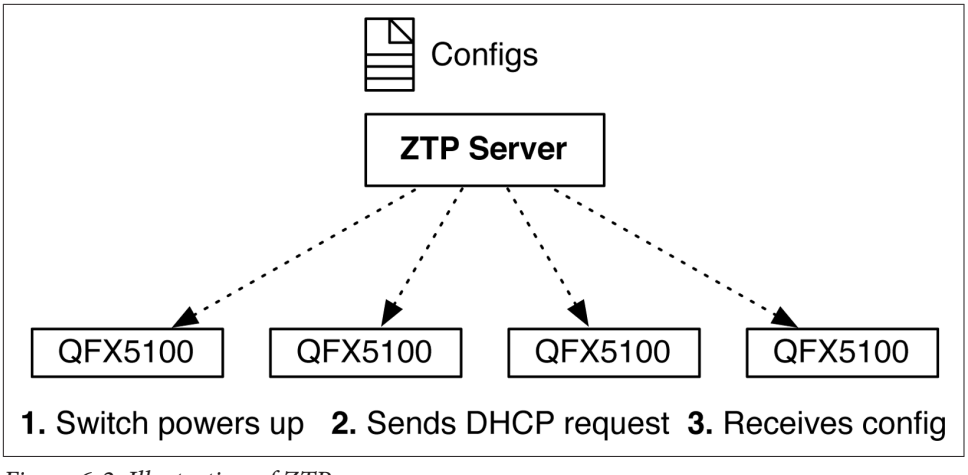


Figure 6-2. Illustration of ZTP

As each Juniper QFX5100 switch powers up, it contacts the ZTP server and requests its configuration. The great benefit of ZTP is that it doesn't matter if you have 10 switches or 100,000; each switch can interact with ZTP in parallel, which means you can quickly bring up an entire data center within minutes. That sure beats logging in to the console of each switch.

The other great benefit of ZTP is that it allows you to automatically upgrade the switch's software in addition to applying a configuration. As you install new switches into the data center, you don't have to worry about software upgrades; ZTP takes care of it all.

ZTP Server

The ZTP server is a simple DHCP server. One of the most popular options for a ZTP server is running a Linux server with the Internet Systems Consortium (ISC) DHCP server, which you can download from the ISC's website. The first step to setting up the ZTP server is to understand what DHCP options you must enable and what they do. Table 6-1 will help out with that task.

Table 6-1. ZTP server DHCP options list

DHCP option	DHCP suboption	Description
07	N/A	Configure one or more syslog servers
12	N/A	Hostname of switch
42	N/A	Configure one or more NTP servers
43	00	Software image filename
43	01	Filename of the configuration file
43	02	Symbolic link flag for software image
43	03	Transfer mode (options include http, ftp, tftp)
43	04	Alternate software image filename
66	N/A	Specify DNS of HTTP/FTP/TFTP host
150	N/A	Specify IP of HTTP/FTP/TFTP host

Being an astute reader, you probably noticed that there are two ways to specify the software image filename; this is done to provide the most portability between different DHCP servers. Some implementations of DHCP do not support DHCP suboption 00, and therefore you must use suboption 04, instead. DHCP suboption 02 might seem a bit confusing, as well. This suboption is a flag that simply informs the DHCP that the filename referenced in suboption 01 is either a real file or a symbolic link to a file. For example, if DHCP suboption 01 points to a symbolic link on a file system, you need to set DHCP suboption 02 to the value "symlink" to indicate that it isn't a real file, but a pointer to a real file. By using a symbolic link, you can always use the same DHCP suboption 01 filename, such as *junos-qfx5100-current.tgz*, and it would always link to the most current software install image, such as *jinstall-qfx-5-13.2X51-D25.2-domestic-signed.tgz*.

When specifying the HTTP/FTP/TFTP server, there are also two DHCP options: 66 and 150. For DHCP servers that support DNS, you can use option 66; otherwise, DHCP option 150 allows you to specify the IP address directly. If both DHCP options 66 and 150 are specified, DHCP option 150 takes precedence.

With DHCP option 43 suboption 03, you can specify the file transfer method. The supported values are "http," "ftp," or "tftp." If DHCP suboption 03 isn't specified, the DHCP server will default to TFTP.

The Juniper QFX5100 also supports additional network automation through DHCP suboption 01, which is traditionally reserved to specify the configuration filename. The Juniper QFX5100 switch downloads this file and then takes a look at the first line in the file to determine its file type. If the line begins with a she-bang (`#!`), the Juniper QFX5100 device will execute the filename as if it were a script. For example, you can use DHCP suboption 01 to specify a Python, shell, or Junos automation script instead

of a traditional configuration file. Imagine all of the possibilities that would be afforded to you by executing a Python script when bootstrapping a switch.

ISC DHCP Configuration

Let's use our new knowledge of the ZTP server DHCP options and begin setting up a new ZTP server. For this laboratory, we'll use the ISC DHCP server, which you can download from the ISC's website. We'll also use the ZTP settings presented in Table 6-2.

Table 6-2. ISC DHCP configuration values

DHCP option and suboption	Value
DHCP option 43, suboption 00	`/jinstall-qfx-5-flex-13.2X51-D20.2-domestic-signed.tgz`
DHCP option 43. suboption 01	`/template.conf`
DHCP option 43, suboption 03	`http`
DHCP option 150	`172.32.32.254`

The first step is to create a `ztp-ops` state to which all of our values can be set. After the state is defined, we'll inform ISC as to what type of value to expect; in this case it's either an IP address or text. The final step is to set up a subnet to accept DHCP requests and associate the `ztp-ops` configuration to this range. The result is that the `dhcpd.conf` looks like this:

```
option ztp-file-server code 150 = { ip-address };
option space ztp-ops;
option ztp-ops.image-file-name code 0 = text;
option ztp-ops.config-file-name code 1 = text;
option ztp-ops.image-file-type code 2 = text;
option ztp-ops.transfer-mode code 3 = text;
option ztp-ops-encap code 43 = encapsulate ztp-ops;

subnet 172.32.32.0 netmask 255.255.255.0 {
  range 172.32.32.20 172.32.32.200;
  option domain-name "provisioning.oob.local";
  option routers 172.32.32.1;
  option broadcast-address 172.32.32.255;
  default-lease-time 600;
  max-lease-time 7200;
  option host-name "netboot";

  option ztp-file-server 172.32.32.254;
  option ztp-ops.image-file-name "/jinstall-qfx-5-flex-13.2X51-D20.2-domestic-
signed.tgz";
  option ztp-ops.transfer-mode "http";
  option ztp-ops.config-file-name "/template.conf";
  }
```

Now, we're ready to handle ZTP requests from any switch in the 172.32.32.0/24 network. Let's use an existing switch in our network that's running an older version of

Junos. You can simulate a factory configuration by using the `request system zero ize` command, which will delete all configuration and cause the switch to reboot.

Take note of the version before zeroing out the switch:

```
dhanks@qfx5100> show version
fpc0:
--------------------------------------------------------------------------
Hostname: qfx5100
Model: qfx5100-48s-6q
JUNOS Base OS Software Suite [13.2X51-D15.5]
```

Note that the Juniper QFX5100 switch is running Junos 13.2X51-D15.5. Next, let's go ahead and zero-out the switch and force it to reboot and come back up in a factory default state. After it reboots, it will perform a DHCP request and the newly configured ZTP server will respond, upgrade the software, and push a new configuration to the switch:

```
dhanks@qfx5100> request system zeroize
warning: System will be rebooted and may not boot without configuration
Erase all data, including configuration and log files? [yes,no] (no) yes

warning: ipsec-key-management subsystem not running - not needed by configuration.
warning: zeroizing fpc0

{master:0}
dhanks@qfx5100> Jul 28 06:42:03 init: chassis-control (PID 35331) stopped by signal
17
Jul 28 06:42:03 init: tnp-process (PID 35329) stopped by signal 17
Terminated
root@temp-leaf-01:RE:0% Jul 28 06:42:09 init: event-processing (PID 977) exited
with status=0 Normal Exit
Waiting (max 60 seconds) for system process `vnlru_mem' to stop...done
Waiting (max 60 seconds) for system process `vnlru' to stop...done
Waiting (max 60 seconds) for system process `bufdaemon' to stop...done
Waiting (max 60 seconds) for system process `syncer' to stop...
Syncing disks, vnodes remaining...0 0 0 0 done

syncing disks... All buffers synced.
Uptime: 5m51s
recorded reboot as normal shutdown
unloading fpga driver
unloading host-dev
Shutting down ACPI
Rebooting...
```

The Juniper QFX5100 switch has rebooted and come back up into a factory default state as shown in the following:

```
Mon Jul 28 06:43:47 UTC 2014

Amnesiac (ttyd0)

login:
```

The next step is that the Juniper QFX5100 switch downloads the software image and configuration file from the ZTP server and reboots again to begin the software installation process:

```
Amnesiac (ttyd0)

login: Terminated
Poweroff for hypervisor to respawn
Jul 28 06:48:54 init: event-processing (PID 1094) exited with status=1
Jul 28 06:48:54 init: packet-forwarding-engine (PID 1357) exited with status=8
Jul 28 06:48:55 init: dhcp-service (PID 1535) exited with status=0 Normal Exit
.
Waiting (max 60 seconds) for system process `vnlru_mem' to stop...done
Waiting (max 60 seconds) for system process `vnlru' to stop...done
Waiting (max 60 seconds) for system process `bufdaemon' to stop...done
Waiting (max 60 seconds) for system process `syncer' to stop...
Syncing disks, vnodes remaining...0 0 0 0 done

syncing disks... All buffers synced.
Uptime: 6m5s
recorded reboot as normal shutdown
unloading fpga driver
unloading host-dev
Powering system off using ACPI
```

Let's check out the logs from the ZTP server to verify that the switch is pulling the correct files:

```
pi@pi /usr/share/nginx/www $ sudo tail -f /var/log/nginx/access.log
172.32.32.176 - - [28/Jul/2014:06:48:48 +0000] "GET //template.conf HTTP/1.1" 200
4919 "-" "fetch libfetch/2.0"
172.32.32.176 - - [28/Jul/2014:06:49:58 +0000] "GET //jinstall-qfx-5-flex-13.2X51-
D20.2-domestic-signed.tgz HTTP/1.1" 200 449262025 "-" "fetch libfetch/2.0"
```

We can see that the Juniper QFX5100 downloaded `/template.conf` first and then 50 seconds later downloaded the new Junos software. The Juniper QFX5100 device has now rebooted and come back online, the new software has been installed, and the new configuration has been applied:

```
Mon Jul 28 06:53:53 UTC 2014

temp-leaf-01 (ttyd0)

login: root
Password:

--- JUNOS 13.2X51-D20.2 built 2014-04-29 08:35:21 UTC
root@temp-leaf-01:RE:0% cli
{master:0}
root@temp-leaf-01> show system uptime
fpc0:
--------------------------------------------------------------------------
Current time: 2014-07-27 23:58:15 PDT
System booted: 2014-07-27 23:50:03 PDT (00:08:12 ago)
Protocols started: 2014-07-27 23:53:56 PDT (00:04:19 ago)
Last configured: 2014-07-27 23:54:18 PDT (00:03:57 ago) by root
11:58PM  up 8 mins, 1 user, load averages: 0.08, 0.64, 0.46
```

```
{master:0}
root@temp-leaf-01> show version
fpc0:
--------------------------------------------------------------------------
Hostname: temp-leaf-01
Model: qfx5100-48s-6q
JUNOS Base OS Software Suite [13.2X51-D20.2]
```

Notice the update of the switch is only 8 minutes and the new software version is Junos 13.2X51-D20.2. Also note that the hostname has changed because the configuration has been applied, as well.

ISC DHCP Review

ZTP is a great way to quickly build the switching fabric of your data center. As soon as the management network is set up, the entire infrastructure required for ZTP is ready. You simply define the standardized Junos software version and configuration, and then you can quickly deploy 1,000s of switches within minutes.

Puppet

When it comes to automating the data center, Puppet represents one of the most common automation products that's used in large-scale deployments. When you need to make changes across a large set of systems, it quickly becomes a burden to do it manually, and automation is required. Puppet is built on abstraction, so you can use it across a large variety of servers and networking equipment.

The Puppet architecture is very simple. Devices that are to be managed are called *Nodes* and the *Puppet Master* acts as the global catalog and change authority for all Nodes. Puppet-specific devices are called Nodes on purpose; this is to completely abstract the device that is being managed. For the exercise in this chapter, the Node shall represent the Juniper QFX5100 switch. The Puppet Node and Master exchange three types of information, as shown in Figure 6-3.

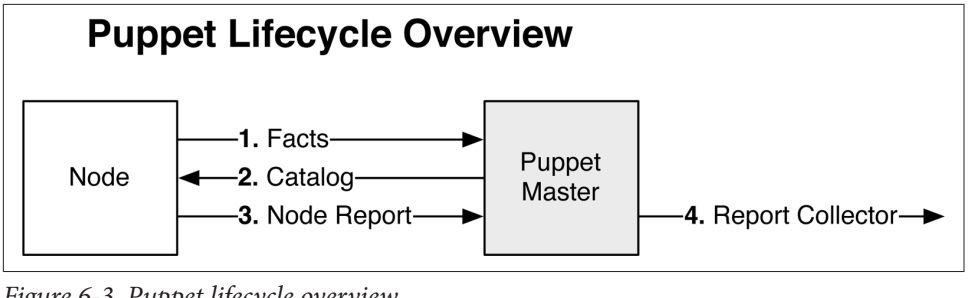


Figure 6-3. Puppet lifecycle overview

The Puppet lifecycle is very simple:

Facts
: The first step is that the Puppet Node reports a list of facts to the Puppet Master. Facts are simply a collection of key/value pairs. In the example (the Juniper QFX5100 switch), a list of facts could include interface names, interface descriptions, and VLAN memberships. The Puppet Node reports a list of facts to the Puppet Master to inform it of the Node's current state. If the current state doesn't match the Puppet Master's catalog, the Node is out-of-date and needs to be updated.

Catalog
: The Puppet Master compiles a catalog based on the facts provided by the Puppet Node. The Puppet Node takes the catalog and applies all changes.

Node Report
: The Puppet Node completes all of the changes specified in the catalog and reports back to the Puppet Master.

Report Collector
: You can use Puppet's open API to send data to third-party collectors and reporting tools to create data center change reports.

So that's Puppet in a nutshell. It's a great way to get started with data center automation, because you can use Puppet across all of your servers, applications, and networking equipment.

Let's take a look at how the Juniper QFX5100 series implements Puppet and turns itself into a Puppet Node that's capable of being managed by the Puppet Master, as depicted in Figure 6-4.

The Juniper QFX5100 family requires that the Juniper SDK JPuppet package be installed before you can use it as a Puppet Node.

Juniper QFX5100 switches also support a new Junos software image called Enhanced Automation that ships preinstalled with the Puppet agent. It's recommended that you use the Enhanced Automation package if you're looking to automate your data center.

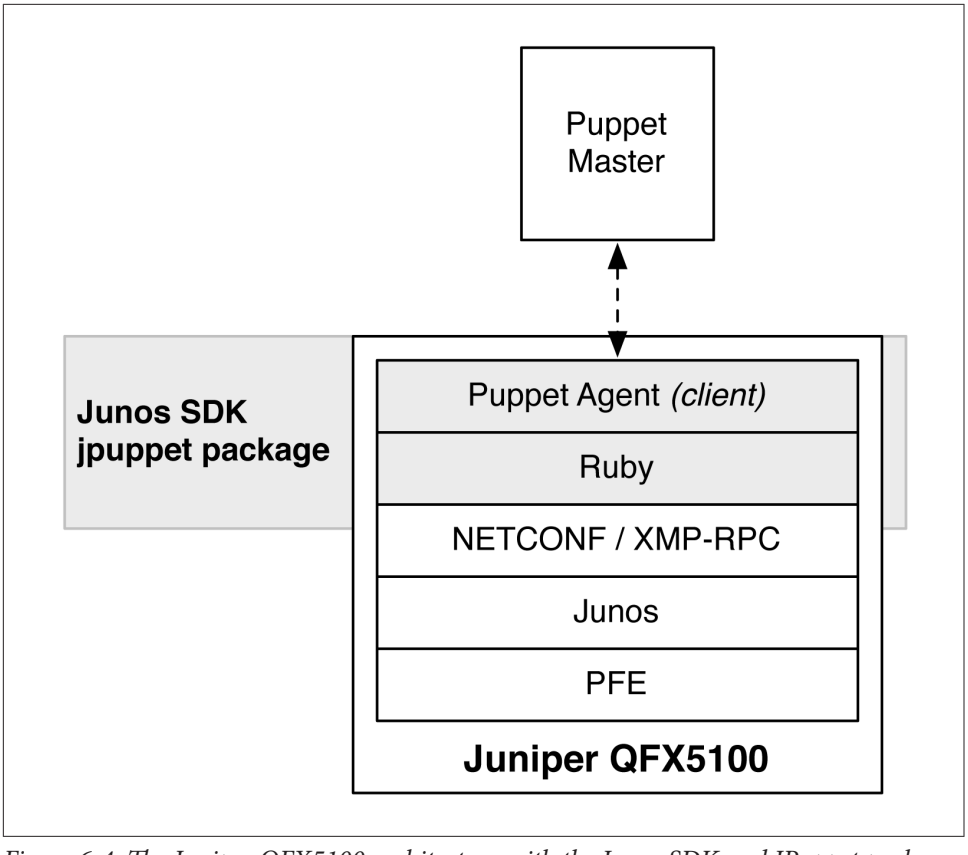


Figure 6-4. The Juniper QFX5100 architecture with the Junos SDK and JPuppet package

The Puppet agent on the Juniper QFX5100 is simply a Ruby daemon that uses the standard NetDev module on the Puppet Master. The Puppet agent is installed into the */opt/sdk/juniper/bin/* directory.

Puppet Agent

The first step in configuring Puppet on the Juniper QFX5100 is to drop into the shell and begin setting up the *puppet.conf* file, as follows:

```
% setenv PATH ${PATH}:/opt/sdk/juniper/bin
```

We'll need to create a couple of directories and start creating our new *puppet.conf*.

```
% mkdir -p $HOME/.puppet/var/run
% mkdir -p $HOME/.puppet/var/log
% vi $HOME/.puppet/puppet.conf
```

The *puppet.conf* file should look something like this:

```
[main]
libdir = $vardir/lib
logdir = $vardir/log/puppet
rundir = $vardir/run/puppet
ssldir = $vardir/ssl
factpath = $libdir/facter
moduledir = $libdir
pluginsync = true

[agent]
server = 172.32.32.254
classfile = $vardir/classes.txt
localconfig = $vardir/localconfig
daemonize = false
```

Be sure to change the `server` to the correct IP address that hosts the Puppet Master. At this point, we should be good to go. The next step is to run the Puppet agent for the first time on the Juniper QFX5100 switch:

```
% puppet agent --test
warning: iconv couldn't be loaded, which is required for UTF-8/UTF-16 conversions
info: Creating a new SSL key for qfx5100

info: Caching certificate for ca
info: Creating a new SSL certificate request for qfx5100
info: Certificate Request fingerprint (md5):
B3:EF:11:56:04:B1:9F:52:C6:4F:46:13:99:BC:B1:5C
err: Could not request certificate: Could not intern from s: header too long
Exiting; failed to retrieve certificate and waitforcert is disabled
%
```

No need to fret; the first time you run the Puppet agent, it creates a new SSL certificate and submits it to the Puppet Master. The Puppet agent cannot continue until the Puppet Master has signed the Secure Sockets Layer (SSL) certificate. Let's log in to the Puppet Master and see what certificates are available for signing:

```
root@puppet-master:~$ puppet cert list
  "qfx5100" (MD5) B3:EF:11:56:04:B1:9F:52:C6:4F:46:13:99:BC:B1:5C
```

We can see the new SSL certificate from the Juniper QFX5100 switch; sign it and give it back to the switch:

```
root@puppet-master:~$ puppet cert sign qfx5100
Notice: Signed certificate request for qfx5100
Notice: Removing file Puppet::SSL::CertificateRequest qfx5100 at
'/var/lib/puppet/ssl/ca/requests/qfx5100.pem'
root@puppet-master:~$
```

Now that you have signed the Juniper QFX5100 certificate, you can go back to the switch and rerun the Puppet agent:

```
% puppet agent --test
info: Retrieving plugin
info: Caching certificate_revocation_list for ca
notice: /File[/var/home/puppet/.puppet/var/lib/puppet]/ensure: created
notice: /File[/var/home/puppet/.puppet/var/lib/puppet/provider]/ensure: created
notice: /File[/var/home/puppet/.puppet/var/lib/puppet/provider/netdev_lag]/ensure:
```

```
...snip...

info: Caching catalog for qfx5100
info: Applying configuration version '1406526155'
info: Creating state file /var/home/puppet/.puppet/var/state/state.yaml
notice: Finished catalog run in 0.11 seconds
%
```

The Puppet agent has successfully run the first time on the switch now that the SSL certificate has been signed by the Puppet Master. There are a few options to invoke on the Puppet agent on the Juniper QFX5100 switch:

Daemon
: If you want to run the Puppet agent as a daemon on the Juniper QFX5100 device, modify the *puppet.conf* on the switch and change the `daemonize` value to `true`. Now, when you execute the command `puppet agent` (without the `--test`), it will automatically turn into a daemon and return you back to the command prompt. By default, it does this every 30 minutes.

Crontab
: You can set up the */etc/crontab* in the FreeBSD shell to execute `/opt/sdk/juniper/bin/puppet agent --onetime` at any interval you wish.

SSH
: If you don't like pulling changes from the Puppet Master, you can use Secure Shell (SSH) to log in to the Juniper QFX5100 and remotely execute the command `/opt/sdk/juniper/bin/puppet agent --onetime` whenever you need.

Ensure that each switch in the network is connected to an NTP so that as the switch insforms the Puppet Master about its facts, the time is synchronized and doesn't cause any issues.

Puppet Master

At this juncture, let's start making some changes to the Juniper QFX5100 switch. All changes in Puppet are defined in the Puppet Master. Let's begin by adding the Juniper QFX5100 device to the Puppet Master site manifest (*site.pp*). We'll add into the manifest the information listed in Table 6-3.

Table 6-3. Puppet Master manifest settings

NetDev object	Key	Value
netdev_device		`$hostname`
netdev_vlan "blue"	`vlan_id`	`100`
netdev_vlan "blue"	`description`	`"the blue VLAN"`
netdev_vlan "green"	`vlan_id`	`200`
netdev_vlan "green"	`description`	`"the green VLAN"`

NetDev object	Key	Value
netdev_vlan "red"	vlan_id	300
netdev_vlan "red"	description	"the red VLAN"
netdev_interface "xe-0/0/14"	untagged_vlan	blue
netdev_interface "xe-0/0/14"	description	"belongs to the blue VLAN"
netdev_interface "xe-0/0/15"	untagged_vlan	green
netdev_interface "xe-0/0/15"	description	"belongs to the green VLAN"
netdev_interface "xe-0/0/16"	untagged_vlan	red
netdev_interface "xe-0/0/16"	description	"belongs to the red VLAN"
netdev_lag "ae0"	ensure	present
netdev_lag "ae0"	active	true
netdev_lag "ae0"	links	xe-0/0/10, xe-0/0/11
netdev_lag "ae0"	lacp	active
netdev_lag "ae0"	minimum_links	1
netdev_interface "ae0"	tagged_vlans	blue, green, red
netdev_interface "ae0"	description	"core to trunk"

The values in Table 6-3 represent a simple setup of a core switch with a tagged interface connecting to the Juniper QFX5100 switch with three VLANs, as shown in Figure 6-5.

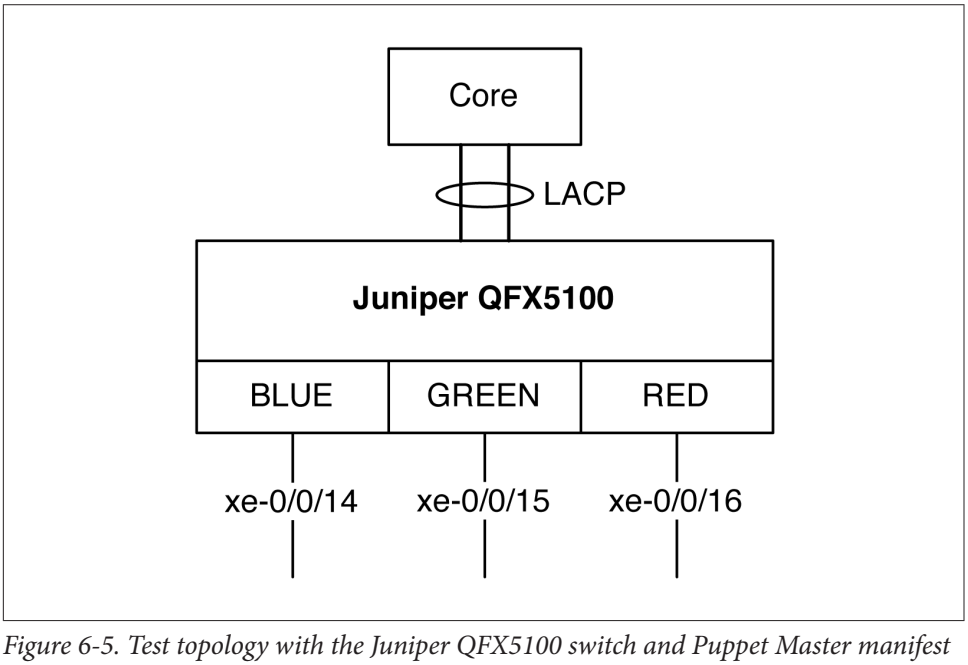


Figure 6-5. Test topology with the Juniper QFX5100 switch and Puppet Master manifest

Now that you understand the topology and what needs to be changed, take the Puppet Master manifest values from Table 6-3 and install them into the *site.pp* on the Puppet Master:

```
node "qfx5100" {
        netdev_device { $hostname: }

        netdev_vlan { "blue":
                vlan_id => 100,
                description => "the blue VLAN",
        }
        netdev_vlan { "green":
                vlan_id => 200,
                description => "the green VLAN"
        }
        netdev_vlan { "red":
                vlan_id => 300,
                description => "the red VLAN",
        }
        netdev_l2_interface { 'xe-0/0/14':
                untagged_vlan => blue,
                description => "belongs to the blue VLAN"
        }
        netdev_l2_interface { 'xe-0/0/15':
                untagged_vlan => green,
                description => "belongs the green VLAN"
        }
        netdev_l2_interface { 'xe-0/0/16':
                untagged_vlan => red,
                description => "belongs to the red VLAN"
        }

        netdev_lag { "ae0":
                ensure => present,
                active => true,
                links => (['xe-0/0/10','xe-0/0/11']),
                lacp => active,
                minimum_links => 1
        }
        netdev_l2_interface { 'ae0':
                tagged_vlans => [ blue, green, red ],
                description => "Trunk to Core"
        }
}
```

With the Puppet Master manifest updated, let's go back to the Juniper QFX5100 swtich and execute the Puppet agent manually to pull the change into the system:

```
% puppet agent --test
info: Retrieving plugin
info: Caching catalog for qfx5100
info: Applying configuration version '1406527872'
notice: /Stage[main]//Node[qfx5100]/Netdev_vlan[blue]/ensure: created
notice: /Stage[main]//Node[qfx5100]/Netdev_vlan[green]/ensure: created
notice: /Stage[main]//Node[qfx5100]/Netdev_lag[ae0]/ensure: created
notice: /Netdev_l2_interface[xe-0/0/15]/ensure: created
notice: /Netdev_l2_interface[xe-0/0/14]/ensure: created
notice: /Stage[main]//Node[qfx5100]/Netdev_vlan[red]/ensure: created
```

```
notice: /Netdev_l2_interface[ae0]/ensure: created
notice: /Netdev_l2_interface[xe-0/0/16]/ensure: created
info: JUNOS: Committing 8 changes.
```

We can see that the Puppet agent has found all of the new netdev components and has committed eight changes. Don't forget that the Puppet agent also sends a report back to the Puppet Master; here is the rest of the output from the `puppet agent --test` command:

```
notice: JUNOS:

[edit interfaces]
+   xe-0/0/10 {
+       ether-options {
+           802.3ad ae0;
+       }
+   }
+   xe-0/0/11 {
+       ether-options {
+           802.3ad ae0;
+       }
+   }
+   xe-0/0/14 {
+       unit 0 {
+           description "belongs to the blue VLAN";
+           family ethernet-switching {
+               interface-mode access;
+               vlan {
+                   members 100;
+               }
+           }
+       }
+   }
+   xe-0/0/15 {
+       unit 0 {
+           description "belongs the green VLAN";
+           family ethernet-switching {
+               interface-mode access;
+               vlan {
+                   members 200;
+               }
+           }
+       }
+   }
+   xe-0/0/16 {
+       unit 0 {
+           description "belongs to the red VLAN";
+           family ethernet-switching {
+               interface-mode access;
+               vlan {
+                   members 300;
+               }
+           }
+       }
+   }
+   ae0 {
+       apply-macro "netdev_lag[:links]" {
+           xe-0/0/10;
```

```
+           xe-0/0/11;
+       }
+       aggregated-ether-options {
+           minimum-links 1;
+           lacp {
+               active;
+           }
+       }
+       unit 0 {
+           description "Trunk to Core";
+           family ethernet-switching {
+               interface-mode trunk;
+               vlan {
+                   members [ 100 200 300 ];
+               }
+           }
+       }
+   }
[edit vlans]
+   blue {
+       description "the blue VLAN";
+       vlan-id 100;
+   }
+   green {
+       description "the green VLAN";
+       vlan-id 200;
+   }
+   red {
+       description "the red VLAN";
+       vlan-id 300;
+   }

notice: JUNOS: OK: COMMIT success!
notice: Finished catalog run in 2.30 seconds
```

The output above is directly from the Junos configuration change control by running the `show compare` command. Each addition is prefixed with a `+` and each deletion is prefixed with a `-` just like the Linux `diff -u` command.

Puppet Review

Puppet is a very powerful data center automation tool for servers and networking devices. Although we only showed the basics, Puppet offers many more features such as using variables and modules to create classes of switches and configure them based upon a certain function such as access switch or core switch.

For more information about Puppet for the Juniper QFX5100 series, visit the Puppet documentation.

For more information about Puppet, go to *http://puppetlabs.com/*.

Chef

The other popular software tool for data center automation is Chef. The Juniper QFX5100 series uses the same architecture for Chef as it does with Puppet, as illustrated in Figure 6-6.

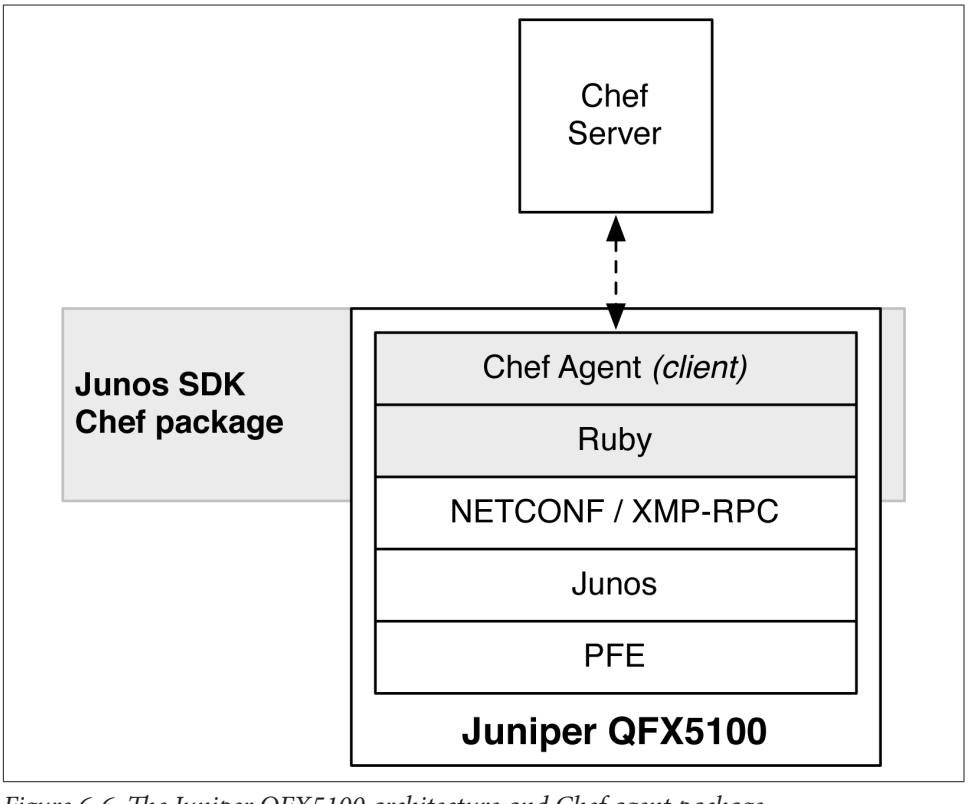


Figure 6-6. The Juniper QFX5100 architecture and Chef agent package

The Chef agent is written as a Ruby program and uses the NETCONF libraries to communicate with Junos. The Junos Enhanced Automation software image comes preinstalled with a Chef agent.

The Chef example we'll use here will be the core switch with a tagged trunk interface going to the Juniper QFX5100 device with three VLANs. To create a working example of Chef with the Juniper QFX5100 switch, we'll use the same data from the Puppet example that's presented in Table 6-3 and Figure 6-5.

Chef Server

The first step is to add the Juniper QFX5100 device to the Chef server. You can choose to do it manually, but Juniper Networks and Chef have created a bootstrap process that takes advantage of Chef automation and makes life much easier. The first step is to pull the bootstrap file from GitHub on our Chef server:

```
root@chef-server:~/chef-repo$ wget https://github.com/opscode/junos-
chef/blob/master/bootstrap/junos-minimal.erb
```

Next, use this bootstrap file as a template and load it into Chef. In this example, our switch IP address is 10.0.0.16:

```
root@chef-server:~/chef-repo$ knife bootstrap 10.0.0.16 --template-file junos-
minimal.erb -x root
Connecting to 10.0.0.16
Password: <Enter the switch's password>
10.0.0.16
10.0.0.16 ---------------------------------------------------------------------
10.0.0.16 -----> Creating required Chef configuration
10.0.0.16 ---------------------------------------------------------------------
10.0.0.16
10.0.0.16 ---------------------------------------------------------------------
10.0.0.16 -----> Performing the initial chef-client run!
10.0.0.16 ---------------------------------------------------------------------
10.0.0.16
10.0.0.16 Starting Chef Client, version 11.10.4
10.0.0.16 Creating a new client identity for qfx5100 using the validator key.
10.0.0.16 Synchronizing Cookbooks:
10.0.0.16 Compiling Cookbooks...
10.0.0.16 Converging 0 resources
10.0.0.16
10.0.0.16 Running handlers:
10.0.0.16 Running handlers complete
10.0.0.16
10.0.0.16 Chef Client finished, 0/0 resources updated in 2.36510424 seconds
root@chef-server:~/chef-repo$
```

When the bootstrap process finishes, you can double-check the Chef server to ensure that you see the Juniper QFX5100 in the client list:

```
root@chef-server:~/chef-repo$ knife client list
chef-validator
chef-webui
qfx5100
```

The Juniper QFX5100 shows up in the client list as expected. To check out additional details, use the following command:

```
root@chef-server:~/chef-repo$ knife node show qfx5100
Node Name:   qfx5100
Environment: _default
FQDN:
IP:
Run List:
Roles:
Recipes:
```

```
Platform:    junos 13.2X51-D21.1
Tags:
root@chef-server:~/chef-repo$
```

Right now, you only have the basic information for the Juniper QFX5100 switch. The next step is to take the netdev data from Table 6-3 and create a Chef recipe. The first step is to create a netdev Chef cookbook.

```
root@chef-server:~/chef-repo$ knife cookbook site download netdev
Downloading netdev from the cookbooks site at version 2.0.0 to /home/root/netdev-
2.0.0.tar.gz
Cookbook saved: /home/root/netdev-2.0.0.tar.gz
root@chef-server:~/chef-repo$ tar zxvf ./netdev-2.0.0.tar.gz -C cookbooks
root@chef-server:~/chef-repo$ mkdir ~/chef-repo/cookbooks/netdev/recipes
```

Now that you have the netdev cookbook and associated directories ready to go, it's time to create some Chef recipes. Let's start with the *~/chef-repo/cookbooks/netdev/recipes/vlan_create.rb* recipe:

```
#
# Cookbook Name:: netdev
# Recipe:: vlan_create
netdev_vlan "blue" do
    vlan_id 100
    description "the blue VLAN"
    action :create
end
netdev_vlan "green" do
    vlan_id 200
    description "the green VLAN"
    action :create
end
netdev_vlan "red" do
    vlan_id 300
    description "the red VLAN"
    action :create
end
```

Now onto the *~/chef-repo/cookbooks/netdev/recipes/access_interface_create.rb* recipe:

```
#
# Cookbook Name:: netdev
# Recipe:: access_interface_create
#
# Physical interface creation using the following defaults:
# auto-negotiation on, MTU 1500, administratively up
netdev_interface "xe-0/0/14" do
    description "access interface"
    action :create
end
netdev_interface "xe-0/0/15" do
    description "access interface"
    action :create
end
netdev_interface "xe-0/0/16" do
    description "access interface"
    action :create
end
# Logical interface creation, setting port mode to access (vlan_tagging false)
```

```
# and assigning interface to a VLAN

netdev_l2_interface "xe-0/0/14" do
    description "belongs to blue VLAN"
    untagged_vlan "blue"
    vlan_tagging false
    action :create
end
netdev_l2_interface "xe-0/0/15" do
    description "belongs to green VLAN"
    untagged_vlan "green"
    vlan_tagging false
    action :create
end
netdev_l2_interface "xe-0/0/16" do
    description "belongs to red VLAN"
    untagged_vlan "red"
    vlan_tagging false
    action :create
end
```

Now onto the *~/chef-repo/cookbooks/netdev/recipes/uplink_interface_create.rb* recipe:

```
#
# Cookbook Name:: netdev
# Recipe:: uplink_interface_create
#
netdev_l2_interface "xe-0/0/10" do
    action :delete
end
netdev_l2_interface "xe-0/0/11" do
    action :delete
end

# Create the LAGs
netdev_lag "ae0" do
    links [ "xe-0/0/10", "xe-0/0/11" ]
    minimum_links 1
    lacp "active"
    action :create
end

# Configure Layer 2 switching on the LAGs. Define the port modeas trunk
# (vlan_tagging true), with membership in the blue, green,and red VLANs.
netdev_l2_interface "ae0" do
    description "Uplink interface"
    tagged_vlans [ "blue", "green", "red" ]
    vlan_tagging true
    action :create
end
```

After you save the three recipes into *~/chef-repo/cookbooks/netdev/recipes/*, you're ready to upload the cookbook into Chef:

```
root@chef-server:~/chef-repo/cookbooks/netdev/recipes$ cd ~/chef-repo/
root@chef-server:~/chef-repo$ knife cookbook upload netdev
Uploading netdev       [2.0.0]
Uploaded 1 cookbook.
```

With the cookbook uploaded, you can associate the recipes with the Juniper QFX5100 switch. You'll need to edit the node by using the following command:

```
root@chef-server:~/chef-repo$ knife node edit qfx5100
Saving updated run_list on node qfx5100
```

Enter the following information into the editor and save it:

```
# Node Runlist
{
    "name": "qfx5100",
    "chef_environment": "_default",
    "normal": {

    },
    "run_list": [
          "recipe[netdev::vlan_create]",
          "recipe[netdev::access_interface_create]",
          "recipe[netdev::uplink_interface_create]"
    ]
}
```

The QFX5100 device is now registered with the three recipes and is ready to create VLANs and assign the access and uplink interfaces.

Chef Agent

For the Juniper QFX5100 switch to pull the new Chef recipes, you'll need to log in to the switch and execute the following command:

```
root@qfx5100:RE:0% /opt/sdk/chef/bin/ruby /opt/sdk/chef/bin/chef-client -c
/var/db/chef/client.rb
Starting Chef Client, version 11.10.4
resolving cookbooks for run list: ["netdev::vlan_create",
"netdev::access_interface_create", "netdev::uplink_interface_create"]
Synchronizing Cookbooks:
  - netdev
Compiling Cookbooks...
Converging 13 resources
Recipe: netdev::vlan_create
  * netdev_vlan[blue] action create
    - create vlan blue with values: vlan_id: 100, description: the blue VLAN

  * netdev_vlan[green] action create
    - create vlan green with values: vlan_id: 200, description: the green VLAN

  * netdev_vlan[red] action create
    - create vlan red with values: vlan_id: 300, description: the red VLAN

Recipe: netdev::access_interface_create
  * netdev_interface[xe-0/0/14] action create
    - create interface xe-0/0/14 with values: description: access interface

  * netdev_interface[xe-0/0/15] action create
    - create interface xe-0/0/15 with values: description: access interface

  * netdev_interface[xe-0/0/16] action create
    - create interface xe-0/0/16 with values: description: access interface
```

```
  * netdev_l2_interface[xe-0/0/14] action create
    - create layer 2 interface xe-0/0/14 with values: vlan_tagging: false,
description: belongs to blue VLAN, untagged_vlan: blue

  * netdev_l2_interface[xe-0/0/15] action create
    - create layer 2 interface xe-0/0/15 with values: vlan_tagging: false,
description: belongs to green VLAN, untagged_vlan: green

  * netdev_l2_interface[xe-0/0/16] action create
    - create layer 2 interface xe-0/0/16 with values: vlan_tagging: false,
description: belongs to red VLAN, untagged_vlan: red

Recipe: netdev::uplink_interface_create
  * netdev_l2_interface[xe-0/0/10] action delete (up to date)
  * netdev_l2_interface[xe-0/0/11] action delete (up to date)
  * netdev_lag[ae0] action create
    - create link aggregation group ae0 with values: links: ["xe-
0/0/10", "xe-0/0/11"], minimum_links: 1, lacp: active

  * netdev_l2_interface[ae0] action create
    - create layer 2 interface ae0 with values: description: Uplink interface,
tagged_vlans: ["blue", "green", "red"], vlan_tagging: true


Running handlers:
  - JunosCommitTransactionHandler
Running handlers complete

Chef Client finished, 11/13 resources updated in 16.496483965 seconds
root@qfx5100:RE:0%
```

The Juniper QFX5100 device successfully downloaded the cookbook and applied the three recipes to the Junos configuration.

Let's log back in to the Chef server and take a look at the Juniper QFX5100 node details and see what has changed:

```
root@chef-server:~/chef-repo$ knife node show qfx5100
Node Name:   qfx5100
Environment: _default
FQDN:
IP:
Run List:    recipe[netdev::vlan_create], recipe[netdev::access_interface_create],
recipe[netdev::uplink_interface_create]
Roles:
Recipes:     netdev::vlan_create, netdev::access_interface_create,
netdev::uplink_interface_create
Platform:    junos 13.2X51-D21.1
Tags:
root@chef-server:~/chef-repo$
```

Much better! Aside from the previous basic information, we can now see that the run list includes all three recipes from the NetDev cookbook.

Just like with Puppet, it's critical that the Juniper QFX5100 and Chef server use NTP to synchronize time so that you don't run into issues when making changes in your data center.

Chef Review

Our Juniper QFX5100 switch has been automated by Chef, utilizing the same configuration data from our previous Puppet laboratory. No matter what your software preference is for automating the data center, the Juniper QFX5100 series helps you quickly deploy changes within seconds. Creating changes is as simple as creating cookbooks and recipes and then applying them to switches in your network.

For more information about how to use Chef with the Juniper QFX5100, visit the Chef documentation.

For more information about Chef, go to *http://www.getchef.com/*.

Junos PyEZ

One of the latest editions to the Juniper QFX5100 network automation toolset is a Python framework called Junos PyEZ. It's designed to provide both programmers and nonprogrammers with the ability to easily automate the Juniper QFX5100 series by using native Python scripting or a simple templating system. You can install Junos PyEZ on any host that supports Python and uses the NETCONF protocol to remotely connect to a Juniper QFX5100 device to make changes or gather data, as shown in Figure 6-7.

Because Junos PyEZ is designed for both programmers and nonprogrammers alike, there are two methods to handle data from Juniper QFX5100 switches:

Structured
: Programmers enjoy using structured data; this simply means that you can take the data received from the Juniper QFX5100 device and load it into native Python data structures, such as lists, sets, and dictionaries.

Unstructured
: There are two types of unstructured data: snippets and templates. The snippets are native Junos output in the forms of text, set, or XML format. The template makes it possible for you to use variables and combine them with the well-known Python Jinja2 templating engine.

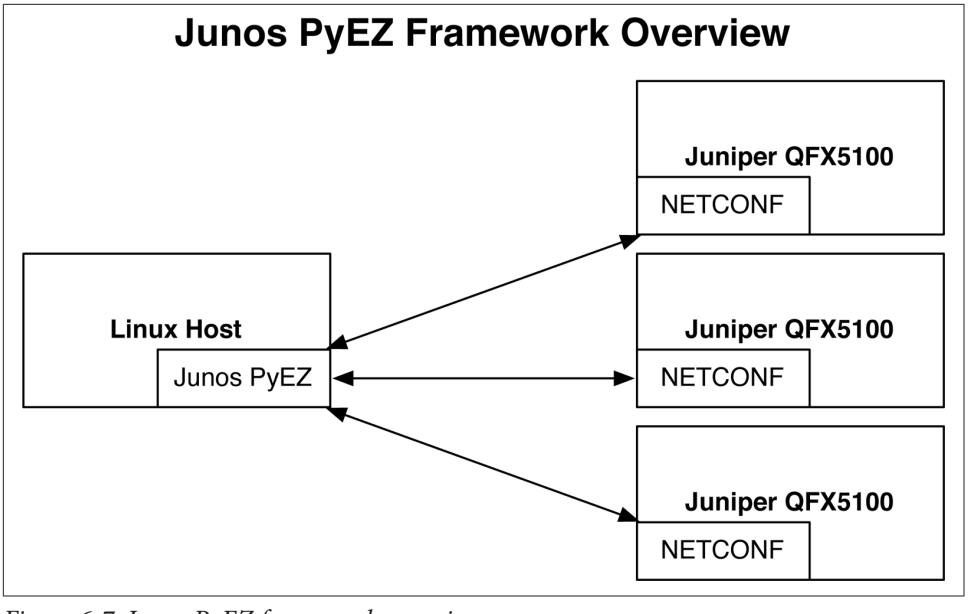


Figure 6-7. Junos PyEZ framework overview

In short, Junos PyEZ allows programmers to use native Python data structures when automating the Juniper QFX5100 family, but also allows nonprogrammers to use a templating system using Jinja2.

The best way to learn Junos PyEZ is to get your hands dirty. Let's get started.

Installation

The installation of Junos PyEZ is straight forward. The best place to install the tools is on any Linux host with IP connectivity to the Juniper QFX5100 switch. The first step is to install a few packages. I'm using an Ubuntu Linux distribution with the APT package manager:

```
root@linux:~$ apt-get install -y python-pip python-dev libxml2-dev
libxslt-dev zlib1g-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

After the prerequisites are installed, you can install the Junos PyEZ package.

```
root@linux:~$ pip install junos-eznc
Downloading/unpacking junos-eznc
  Downloading junos-eznc-1.0.0.tar.gz (54kB): 54kB downloaded
  Running setup.py (path:/tmp/pip_build_root/junos-eznc/setup.py) egg_info for
package junos-eznc
```

That's it; just two commands. Now let's get started with our first script.

Hello, World!

As the programming gods dictate, we must start with a traditional Hello, world! Open your favorite text editor and create the following *helloEZ.py* file:

```
#!/usr/bin/python
from jnpr.junos import Device
dev = Device( user='netconf-test', host='lab-switch', password='lab123' )
dev.open()
print dev.facts
dev.close()
```

The first step is to import the `Device` class from the `jnpr.junos` library. Now, you can make a new `Device` called `dev`; the only requirement is that you specify the username, hostname, and a password to which to connect.

The next step is to simply open a connection to the device, get the facts, and then close the connection. Take a look at the output and see what's included in the facts:

```
root@linux:~/qfxbook$ python ./helloEZ.py
{'domain': None, 'hostname': 'qfx5100', 'ifd_style': 'SWITCH', 'version_info':
junos.version_info(major=(13, 2), type=X, minor=(51, 'D', 15), build=5),
'version_RE0': '13.2X51-D15.5', '2RE': True, 'serialnumber': 'VB3714190366',
'fqdn': 'qfx5100', 'switch_style': 'VLAN', 'version': '13.2X51-D15.5', 'master':
'RE0', 'HOME': '/var/home/netconf-test', 'model': 'QFX5100-96S-8Q', 'RE0':
{'status': 'OK', 'last_reboot_reason': '0x400:bios auto recovery reset ', 'model':
'QFX Routing Engine', 'up_time': '2 hours, 6 minutes, 29 seconds',
'mastership_state': 'master'}, 'personality': 'SWITCH'}
```

The output from the script is returned in a native Python data structure called a dictionary; these are simple key/value pairs. The facts pertaining to the Juniper QFX5100 switch include basic elements such as the hostname, version, and uptime, as shown in the preceding output.

Configuration Management

Using the same laboratory from the Chef and Puppet examples, let's use the Junos PyEZ library to provision the same configuration changes in Table 6-3 and Figure 6-5. To accomplish this, we'll use the Jinja2 templating system. We'll need to create the following files to accomplish this:

demo-template.yml
: The *demo-template.yml* file is a simple template that contains a series of variables to represent a list of the three VLANs and their associated interfaces.

demo-template.j2
: The *demo-template.j2* file is the Jinja2 template file that's used to take the variable input from *demo-template.yml* and create a Junos-formatted configuration file using the variables from *demo-template.yml*.

demo.py
: This is the Python code that employs the Jinja2 templating system and uses *demo-template.yml* and *demo-template.j2* to import the VLAN and interface data and create a Junos configuration file that can be applied to the Juniper QFX5100 device.

Let's get started. The first step it to create the *demo-template.yml* file that contains the following data:

```
---
vlans:
  blue:
    vlan_id: 100
    desc: "the BLUE vlan"
    interfaces:
      - name: xe-0/0/14
  green:
    vlan_id: 200
    desc: "the GREEN vlan"
    interfaces:
      - name: xe-0/0/15
  red:
    vlan_id: 300
    desc: "the RED vlan"
    interfaces:
      - name: xe-0/0/16
uplinks:
  ae0:
    interfaces:
      - name: xe-0/0/10
      - name: xe-0/0/11
    min_links: 1
    lacp_mode: active
```

Next, create the *demo-template.j2* file that contains the following data:

```
vlans {

}

interfaces {



}
```

The final step is to create the Python script that utilizes both of the demo-template files and connects to the Juniper QFX5100 switch, making the appropriate changes. Create the following *demo.py* file:

```
from jnpr.junos.utils.config import Config
from jnpr.junos import Device
import yaml

dev = Device( user='netconf-test', host='lab-switch', password='lab123' )
dev.open()
```

```
dev.bind(cu=Config)
dev.cu

tvars = yaml.load(open("demo-template.yml").read())
dev.cu.load(template_path="demo-template.j2", template_vars=tvars, format="text")

commit_diff = dev.cu.diff()
print commit_diff

dev.cu.commit()

dev.close()
```

As you can see, the *demo.py* script looks similar to the initial *helloPY.py* script that we created earlier. You begin by defining a class for the Juniper QFX5100 switch and opening a connection to it. The next step is to open the *demo-template.yml* file, which contains the simple VLAN and interface variables, and apply it to the Jinja2 templating system. Now, you print the delta between the current running configuration and the candidate configuration loaded by Jinja2. Finally, commit the changes to the Juniper QFX5100 device and close the connection.

The idea behind the two demo-template files is that a user only needs to modify the simple *demo-template.yml* file to make changes to a switch. The *demo-template.j2* file is only a definition file that applies the data from *demo-template.yml* and makes sure the output is in a Junos configuration format.

For more information about PyEZ and templates, visit *https://github.com/Juniper/community-NCE*.

Operational Automation

You can also use Junos PyEZ to operationally interact with the Juniper QFX5100 series without having to make configuration changes. For example, you can create a custom Python script that queries the Juniper QFX5100 device and only returns the interface flap information. Let's give it a shot. Create the following *port-report.py* file:

```
from jnpr.junos.op.phyport import *
from jnpr.junos import Device

dev = Device( user='netconf-test', host='lab-switch', password='lab123' )
dev.open()

ports = PhyPortTable(dev).get()
print "Port,Status,Flapped" #Print Header for CSV

for port in ports:
        print("%s,%s,%s" % (port.key, port.oper, port.flapped))
```

Just as before, you create a `Device` class, give it the login information for the Juniper QFX5100 switch, and then open a NETCONF session to the switch. The next step is to get information about the physical interfaces on the Juniper QFX5100 by using the `PhyPortTable(dev).get()` function and assign it to the `ports` variable.

You can now can loop through the `ports` variable and print the interface flap information for each port. Take a look at the output of the script:

```
root@linux:~$ python ./port-report.py
Port,Status,Flapped
ge-0/0/12,up,2012-01-01 00:25:32 UTC (00:04:27 ago)
ge-0/0/14,up,2012-01-01 00:25:32 UTC (00:04:28 ago)
ge-0/0/16,up,2012-01-01 00:25:42 UTC (00:04:18 ago)
ge-0/0/18,up,2012-01-01 00:25:42 UTC (00:04:18 ago)
ge-0/0/47,down,2012-01-01 00:03:05 UTC (00:26:55 ago)
```

Easy peasy! You can imagine how easy this would be to do across a set of 1,000 switches. We could simply loop through a list of IP addresses and globally see the interface flap information across the entire data center. Any type of information you need to get access to that isn't natively provided by Junos can now be quickly programmed by using the Junos PyEZ framework.

Further Reading

The PyEZ library is under constant development by Juniper Networks and members of the community. At of this writing, the best places for up-to-date information on the project is over on the project's GitHub page. You can also find active library documentation drawn directly from the code itself on the ReadTheDocs website.

To engage with our active project community, be sure to join our Google Group. We would like to hear feedback from people using our PyEZ library in the form of feature requests, but also by active development from the community. We regularly take GitHub "Pull Requests" from the community, and community members have contributed several key pieces of the project.

The author has also setup a GitHub repository for configurations and other bonus material. Please visit *https://github.com/Juniper/qfx5100-book* for more information.

Summary

In this chapter we covered the highlights of the network automation that's available on the Juniper QFX5100 family of switches. We first reviewed the architecture of the Juniper QFX5100 series and how all of the components take advantage of the NETCONF/DMI interfaces to enable network automation. We then took a look at how to quickly get the switch up and running from a factory default configuration by using the ZTP feature. We configured a ZTP server via the vendor DHCP options and were able to quickly upgrade the software of the switch as well as install a new configuration.

Next, we took a look at how to use Puppet and Chef with the Juniper QFX5100 family. Puppet and Chef represent the most popular IT automation tools in the industry; you can use these tools across a wide variety of servers and networking equipment.

The end result is that you can use the same IT automation tool when deploying applications and servers to ensure that the network is set up correctly, as well.

We explored the Junos PyEZ framework and created our first Hello, world! script. We then took the previous laboratory data from the Chef and Puppet sections and provisioned the same VLAN and interface configuration changes by using the Junos PyEZ framework. Finally we showed a simple operational script that looks at the interface flap information on a Juniper QFX5100 switch. Using the Junos PyEZ framework is a great way to quickly gain access to the information you need to operate a data center.

Network automation in the data center is an important topic, and the Juniper QFX5100 series delivers in spades. From ZTP to Junos Enhanced Automation, the Juniper QFX5100 supports all of the major programming languages:

- Python
- Go
- Ruby
- Perl

Juniper Networks has invested heavily in the open source community with the Puppet and Chef NetDev framework and the Junos PyEZ framework. Do you have a project that could benefit from network automation? Go get busy!

CHAPTER 7

IP Fabrics (Clos)

Everywhere you look in the networking world you see something about IP Fabrics or Clos networks. Something is brewing, but what is it? What's driving the need for IP Fabrics? If an IP Fabric is the answer, then what is the problem we're trying to solve?

Many over-the-top (OTT) and software-as-a-service (SaaS) companies have been building large IP Fabrics for a long time, but have rarely received any attention for having done so. Such companies generally have no need for compute virtualization and write their applications in such a way that high availability is built in to the application. With intelligent applications and no compute virtualization, it makes a lot of sense to build an IP Fabric using nothing but Layer 3 protocols. Layer 2 has traditionally been a point of weakness in data centers with respect to scale and high availability. It's a difficult problem to solve when you have to flood traffic across a large set of devices and prevent loops on Ethernet frames that don't natively have a time-to-live field.

If companies have been building large IP Fabrics for a long time, why is it that only recently IP Fabrics have been receiving a lot of attention? The answer is because of overlay networking in the data center. The problem being solved is twofold: first, network agility, and second, simplifying the network. Overlay networking combined with IP Fabrics in the data center is an interesting way of providing both agility and simplifying the provisioning of data center resources.

Overlay Networking

One of the first design considerations in a next-generation data center is do you need to centrally orchestrate all resources within it such that you can deploy applications within seconds? The follow-up question is do you currently virtualize your data center compute and storage with hypervisors or cloud management platforms? If the

answer is "yes" to these questions, you must consider an overlay architecture when it comes to the data center network.

Given that compute and storage have already been virtualized, the next step is to virtualize the data center network. Using an overlay architecture in the data center gives you the freedom to decouple physical hardware from the network, which is one of the key tenets of virtualization. Decoupling the network from the physical hardware makes it possible for the data center network to be programmatically provisioned within seconds

The second benefit of overlay networking is that it supports both Layer 2 and Layer 3 transport between virtual machines (VMs) and servers, which is very compelling to traditional IT data centers. The third benefit is that overlay networking has a much larger scale than traditional Virtual Local Area Networks (VLANs) and supports up to 16.7 million tenants. Two great examples of products that support overlay architectures are Juniper Contrail and VMware NSX.

Moving to an overlay architecture places a different "network tax" on the data center. Typically, when servers and virtual machines are connected to a network, they each consume a MAC address and host route entry in the network. However, in an overlay architecture, only the Virtual Tunnel End Points (VTEPs) consume a MAC address and host route entry in the network. All VM and server traffic is now encapsulated between VTEPs and the MAC address, and the host route of each VM and server isn't visible to the underlying networking equipment. The MAC address and host route scale have been moved from the physical network hardware into the hypervisor.

Bare-Metal Servers

It's rare to find a data center that has virtualized 100 percent of its compute resources. There's always a subset of servers that cannot be virtualized due to performance, compliance, or any other number of reasons. This raises an interesting question: if 80 percent of the servers in the data center are virtualized and take advantage of an overlay architecture, how do you provide connectivity to the other 20 percent?

Overlay architectures support several mechanisms to provide connectivity to physical servers. The most common option is to embed a VTEP into the physical access switch, as shown in Figure 7-1.

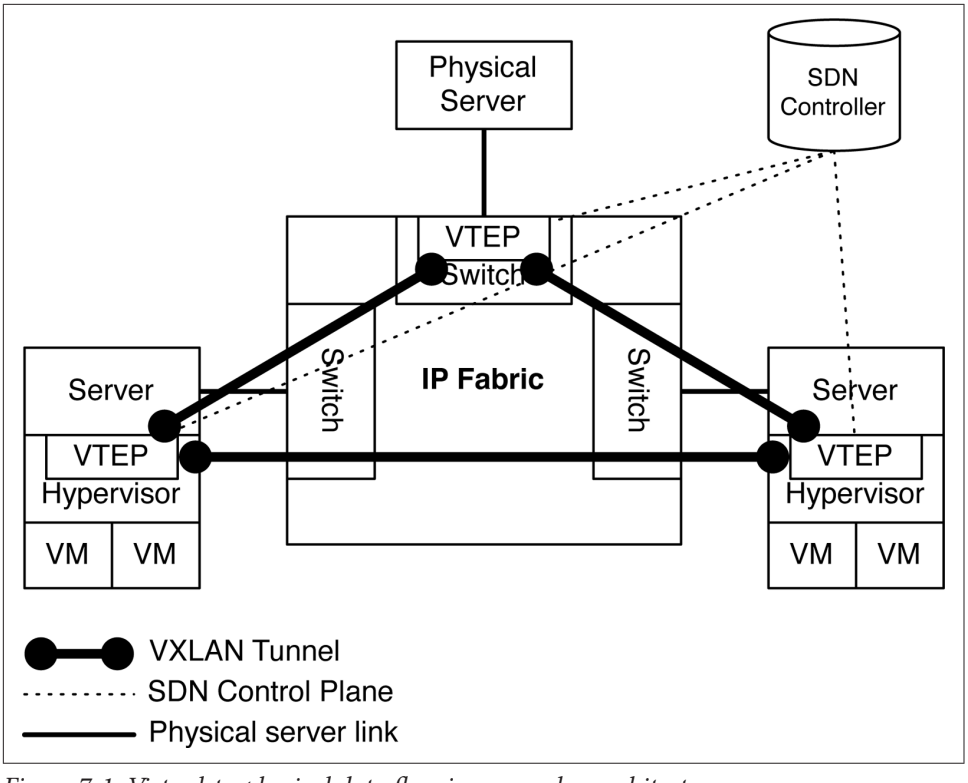


Figure 7-1. Virtual-to-physical data flow in an overlay architecture

Each server on the left and right of the IP Fabric in Figure 7-1 has been virtualized with a hypervisor. Each hypervisor has a VTEP within it that handles the encapsulation of data plane traffic between VMs. Each VTEP also handles MAC address learning, provisioning of new virtual networks, and other configuration changes. The server on top of the IP Fabric is a simple physical server, but doesn't have any VTEP capabilities of its own. For the physical server to participate in the overlay architecture, it needs something to encapsulate the data plane traffic and perform MAC address learning. Being able to handle the VTEP role within an access switch simplifies the overlay architecture. Now, each access switch that has physical servers connected to it can simply perform the overlay encapsulation and control plane on behalf of the physical server. From the point of view of the physical server, it simply sends traffic into the network without having to worry about anything else.

IP Fabric

To summarize, there are two primary drivers for an IP Fabric: OTT companies with simple Layer 3 requirements, and the introduction of overlay networking that uses

the IP Fabric as a foundational underlay. Let's begin to explore these by taking a look at the requirements of overlay networking in the data center and how an IP Fabric can meet and exceed the requirements.

All VM and server MAC addresses, traffic, and flooding are encapsulated between VTEPs in an overlay architecture. The only network requirements of VTEPs is Layer 3 connectivity. Creating a network that's able to meet the networking requirements is straightforward. The challenge is in how you design a transport architecture that's able to scale in a linear fashion as the size increases. A very similar problem was solved back in 1953, by the telecommunications industry. Charles Clos invented a method to create a multistage network that is able to grow beyond the largest switch in the network. This is illustrated in Figure 7-2.

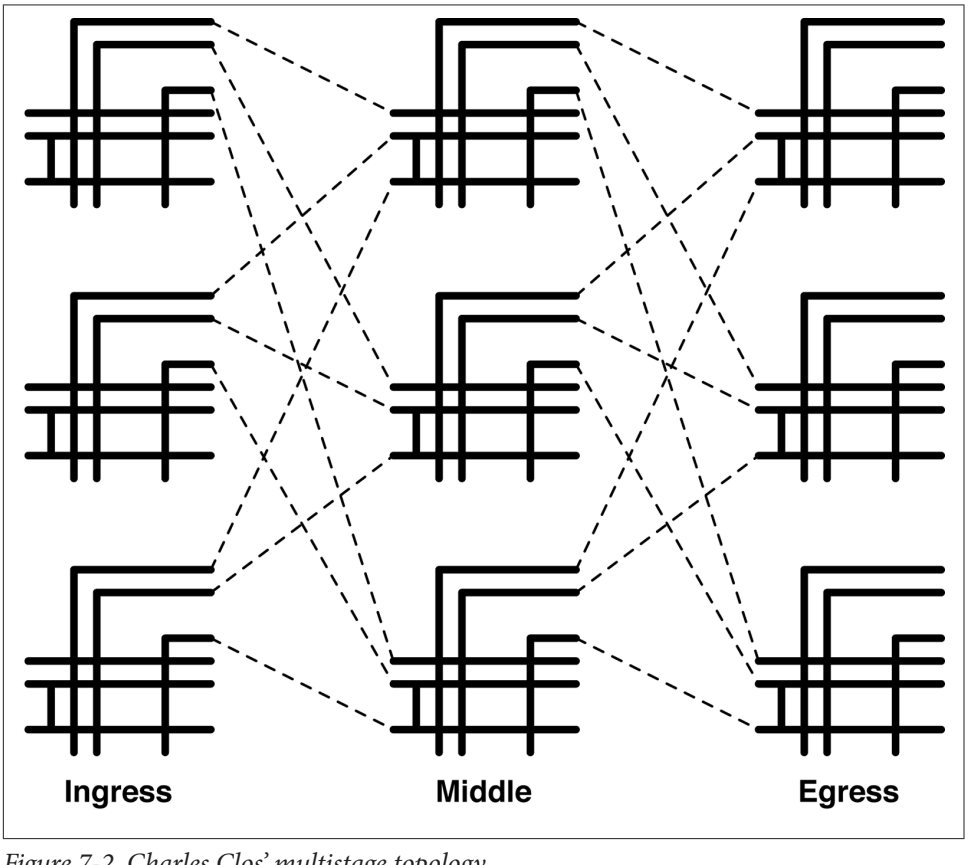


Figure 7-2. Charles Clos' multistage topology

The advantage to a Clos topology is that it's nonblocking and provides predictable performance and scaling characteristics. Figure 7-2 represents a 3-stage Clos network: ingress, middle, and egress.

We can take the same principals of the Clos network and apply it to creating an IP Fabric. Many networks are already designed like this and are often referred to as spine-and-leaf networks, which you can see demonstrated in Figure 7-3.

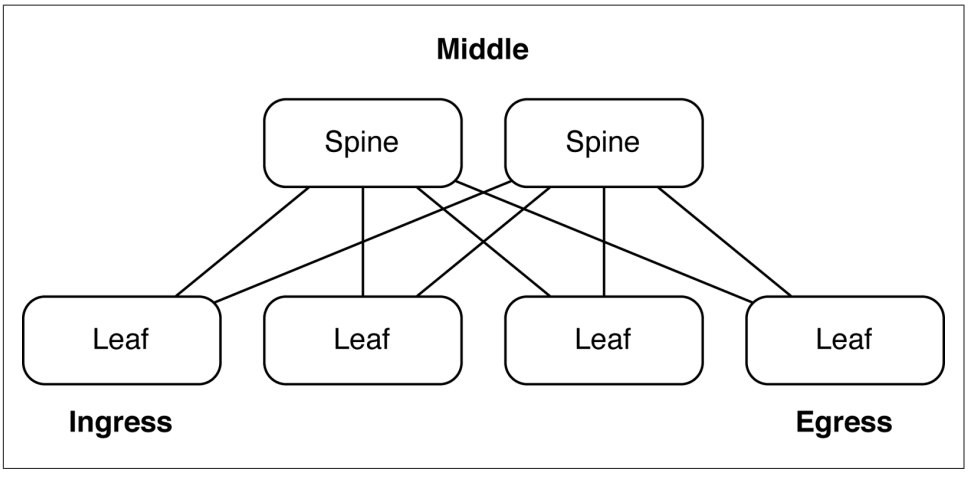


Figure 7-3. Spine-and-leaf topology

A spine-and-leaf network is actually identical to a three-stage Clos network; it is sometimes referred to as a *folded* three-stage Clos network because the ingress and egress points are folded back on top of each other, as illustrated in Figure 7-3. In this example the spine switches are simple Layer 3 switches and the leaves are top-of-rack (ToR) switches that provide connectivity to the servers and VTEPs.

The secret to scaling up the number of ports in a Clos network is adjusting two values: the width of the spine, and over-subscription ratio. The wider the spine, the more leaves the IP Fabric can support. The more over-subscription placed into the leaves, the larger the IP Fabric, as well. Let's review some example topologies in detail to understand how the pieces are put together and what the end results are.

768×10GbE Virtual Chassis Fabric

The first example is a new Juniper technology called Virtual Chassis Fabric (VCF) that enables you to create a three-stage IP Fabric by using a set of Juniper QFX5100 switches that are managed as a single device. As of Junos 13.2, the maximum number of switches in a VCF is 20, as depicted in Figure 7-4.

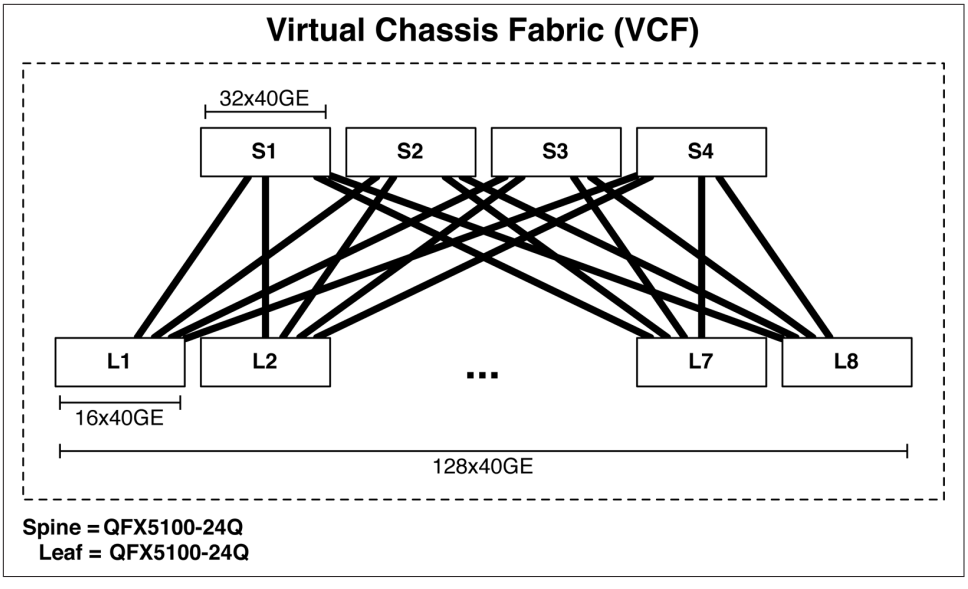


Figure 7-4. VCF of 768 10GbE ports

In this example, the spine comprises four QFX5100-24Q switches; each switch supports up to 32 40GbE interfaces. The leaves are built by using the Juniper QFX5100-48S, which supports 48 10GbE and 6 40GbE interfaces. Each leaf uses 4 40GbE interfaces as uplinks, with one link going to each spine (see Figure 7-4); this creates an over-subscription of 480:160, or 3:1 per leaf. Because VCF only supports 20 switches, we have a total of four spine switches and 16 leaf switches for a total of 20. Each leaf supports 48 10GbE interfaces; because there are 16 leaves total, this brings the total port count up to 768 10GbE, with 3:1 over-subscription.

If you have scaling requirements that exceed the capacity of VCF, it's not a problem. The next option is to create a simple three-stage IP Fabric that is able to scale to thousands of ports.

3,072×10GbE IP Fabric

The next option is creating a simple three-stage IP Fabric by using the Juniper QFX5100-24Q and QFX5100-96S, but this time we won't use VCF. The Juniper QFX5100-24Q switch has 32 40GbE ports, and the Juniper QFX5100-96S boasts 96 10GbE and 8 40GbE ports. Combining the Juniper QFX5100-24Q and the Juniper QFX5100-96S creates an IP Fabric of 3,072 usable 10GbE ports, as shown in Figure 7-5.

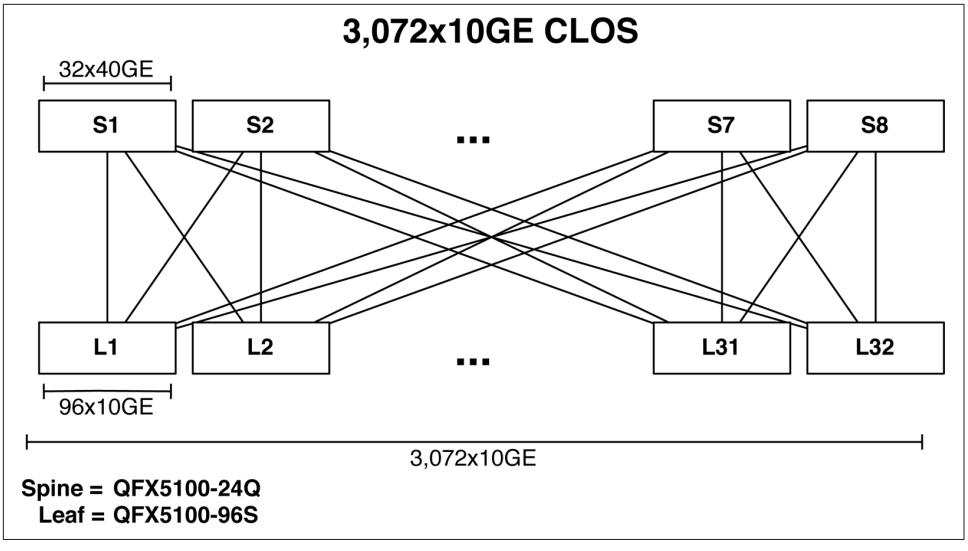


Figure 7-5. 3,072 10GbE IP Fabric topology

The leaves are constructed by using the Juniper QFX5100-96S, and 8 40GbE interfaces are used as uplinks into the spine. Because each leaf has eight uplinks into the spine, the maximum width of the spine is eight. Each 40GbE interface per leaf will connect to a separate spine; thus each leaf will consume one 40GbE interface per spine. To calculate the maximum size of the IP Fabric, you need to multiply the number of server interfaces on the leaf by the number of leaves supported by the spine. In this example, the spine can support 32 leaves, and each leaf can support 96 ports of 10GbE; this is a total of 3,072 usable 10GbE ports with a 3:1 over-subscription ratio.

Control Plane Options

One of the big benefits to using VCF is that you need not worry about the underlying control plane protocols of the IP Fabric. It just works. However, if you need to create a network that exceeds the scale of VCF, you need to take a look at what the control plane options are.

One of the fundamental requirements in creating an IP Fabric is the distribution of prefixes. Each leaf will need to send and receive IP routing information to and from all of the other leaves in the IP Fabric. The question now becomes what are the options for an IP Fabric control plane, and which is the best? We can begin by reviewing the fundamental requirements of an IP Fabric and mapping the results to the control plane options, as is done in Table 7-1.

Table 7-1. IP Fabric requirements and control plane options

Requirement	OSPF	IS-IS	BGP
Advertise prefixes	Yes	Yes	Yes
Scale	Limited	Limited	Extensive
Traffic engineering	Limited	Limited	Extensive
Traffic tagging	Limited	Limited	Extensive
Multivendor stability	Yes	Yes	Extensive

The most common options for the control plane of an IP Fabric are Open Shortest Path First (OSPF), Intermediate System to Intermediate System (IS-IS), and Border Gateway Protocol (BGP). Each protocol can fundamentally advertise prefixes, but vary in terms of scale and features. OSPF and IS-IS use a flooding technique to send updates and other routing information. Creating areas can help scope the amount of flooding, but then you start to lose the benefits of a Shortest Path First (SPF) routing protocol. On the other hand, BGP was created from the ground up to support a large number of prefixes and peering points. The best use case in the world to prove this point is the Internet.

Having the ability to shift traffic around in an IP Fabric could be useful; for example, you could steer traffic around a specific spine switch while it's in maintenance. OSPF and IS-IS have limited traffic-engineering and traffic-tagging capabilities. Again, BGP was designed from the ground up to support extensive traffic engineering and tagging with features such as Local Preference, Media Endpoint Discoveries (MEDs), and extended communities.

One of the interesting side effects of building a large IP Fabric is that it's generally done iteratively and over time. It is common to see multiple vendors creating a single IP Fabric. Although OSPF and IS-IS work well across multiple vendors, the real winner here is BGP. Again, the best use case in the world is the Internet. It consists of a huge number of vendors, equipment, and other variables, but they all use BGP as the control plane protocol to advertise prefixes, perform traffic engineering, and tag traffic.

Because of the scale, traffic tagging, and multivendor stability, BGP is the best choice when selecting a control plane protocol for an IP Fabric. The next question is how do you design BGP in an IP Fabric?

BGP Design

One of the first decisions to make is to whether to use iBGP or eBGP. The very nature of an IP Fabric is based on Equal-Cost Multipath (ECMP). One of the design considerations is how does each option handle ECMP? By default eBGP supports ECMP

without a problem. However, iBGP requires a BGP route reflector and the AddPath feature to fully support ECMP.

Let's take a closer look at the eBGP design in an IP Fabric. Each switch represents a different autonomous system (AS) number and each leaf must peer with every other spine in the IP Fabric, as illustrated in Figure 7-6.

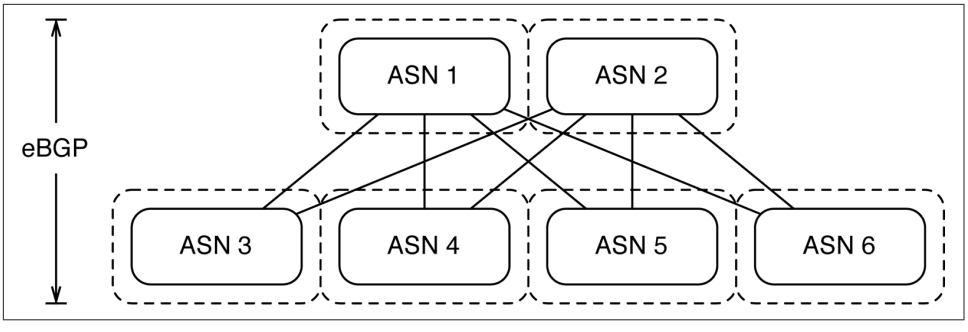


Figure 7-6. Using eBGP in an IP Fabric

Using eBGP in an IP Fabric is very simple and straightforward; it also lends itself well to traffic engineering using Local Preference and AS padding techniques.

Designing iBGP in an IP Fabric is a bit different, due to the requirements of iBGP to have all switches peer with every other device within the IP Fabric. To mitigate the burden of having to peer with every other device in the IP Fabric, we can use inline BGP route reflectors in the spine of the network (see Figure 7-7). The problem with standard BGP route reflection is that it only reflects the best prefix and doesn't lend itself well to ECMP. To enable full ECMP we must use the BGP AddPath feature, which adds additional ECMP paths into the BGP advertisements between the route reflector and clients.

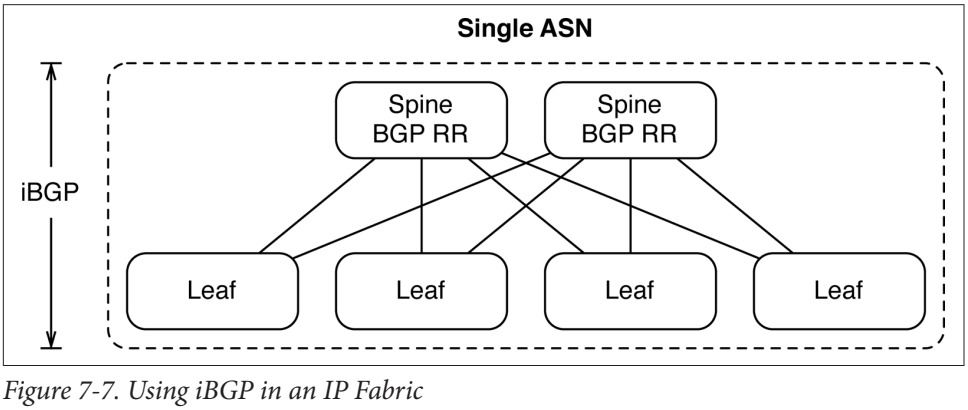


Figure 7-7. Using iBGP in an IP Fabric

The Juniper QFX5100 series supports both the iBGP and eBGP design options. Both options work equally well; however, the design and implementation of eBGP is simpler. Going back to designing a machine with less moving parts is always more stable; it's our recommendation to use eBGP when creating simple three-stage IP Fabrics. There's no need to worry about BGP route reflection and AddPath if you're not required to do so.

Implementation Requirements

There is a set of requirements that need to be worked out in order to create a blueprint for an IP Fabric. At a high level, it revolves around IP Address Management (IPAM) and BGP assignments. The list that follows breaks out the requirements into the next level of detail:

Base IP Prefix
: All of the IP address assignments made within the IP Fabric must originate from a common base IP prefix. It's critical that the base IP prefix have enough address space to hold all of the point-to-point addressing as well as loopback addressing of each switch in the IP Fabric.

Point-to-Point Network Mask
: Each leaf is connected to every spine in the IP Fabric; these connections are referred to as the point-to-point links. The network mask used in the point-to-point links will determine how much of the base IP prefix is used. For example, using a 30-bit network mask will use twice as much space as using a 31-bit network mask.

Point-to-Point IP Addresses
: For every point-to-point connection, each switch must have an IP address assignment. You need to decide whether the spine receives the lower or higher numbered IP address assignment. This is more of a cosmetic decision and doesn't impact the functionality of the IP Fabric.

Server-Facing IP Prefix
: To provide Layer 3 gateway services to VTEPs, the leaves must have a consistent IP prefix that's used for server-facing traffic. This is separate from the base IP prefix used to construct the IP Fabric. The server-facing IP prefix must be large enough to support the address requirements of each leaf in the IP Fabric. For example, if each leaf required a 24-bit subnet and there were 512 leaves, the minimum server-facing IP prefix would need to be at least 15 bits, such as 192.168.0.0/15, which would allow you to have 512 24-bit subnets. Each leaf would have a 24-bit subnet such as 192.168.0.0/24 and could use the first IP address for Layer 2 gateway services such as 192.168.0.1/24.

Loopback Addressing
: Each switch in the IP Fabric needs a single loopback address using a 32-bit mask. You can use the loopback address for troubleshooting and to verify connectivity between switches.

BGP Autonomous System Numbers
: Each switch in the IP Fabric would require its own Autonomous System Numbers (ASN). Each spine and each leaf would have a unique BGP ASN. This would make it possible for eBGP to be used between the leaves and spines.

BGP Export Policy
: Each of the leaves needs to advertise its local server-facing IP prefix into the IP Fabric so that all other servers know how to reach it. Each leaf would also need to export its loopback address into the IP Fabric, as well.

BGP Import Policy
: Because each leaf only cares about server-facing IP prefixes and loopback addressing, all other addressing of point-to-point links can be filtered out.

Equal Cost Multi-Path Routing
: Each spine and leaf should have the ability to load balance flows across a set of equal next-hops. For example, if there are four spine switches, each leaf would have a connection to each spine. For every flow egressing a leaf switch, there should exist four equal next-hops: one for each spine. To do this ECMP routing should be enabled.

These requirements can easily build a blueprint for the IP Fabric. Although the network might not be fully built out from day one, it's good to have a scaled out blueprint of the IP Fabric so that there's no question on how to scale out the network in the future.

Decision Points

There are a couple of important decision points when designing an IP Fabric. The first decision is whether to use iBGP or eBGP. At first this might seem like a simple choice, but there are some other variables that make the decision a bit more complicated. The second decision point is actually a fallout of the first: should you use 16-bit or 32-bit ASNs? Let's walk through the decision points one at a time and take a closer look.

The first decision point should take you back to your JNCIE or CCIE days. What are the requirements of iBGP versus eBGP? We all know that iBGP requires a full mesh to propagate prefixes throughout the topology. However, eBGP doesn't require a full mesh and is more flexible. Obviously, the reason behind this is loop prevention. To prevent loops, iBGP will not propagate prefixes learned from one iBGP peer to another. Each iBGP switch must have a BGP session to the other to fully propagate

routes. On the other hand, eBGP will simply propagate all BGP prefixes to all BGP neighbors; the exception is that any prefixes that contain the switch's own ASN will be dropped.

iBGP design

Let's focus on an iBGP design that meets all of the implementation requirements of building an IP Fabric. The first challenge is how do you get around the full mesh requirement of iBGP? The answer will be BGP confederations or route reflection. Given that an IP Fabric is a fixed topology, route reflection lends itself nicely. As demonstrated in Figure 7-8, each spine switch can act as a BGP route reflector, whereas each leaf is a BGP route reflector client.

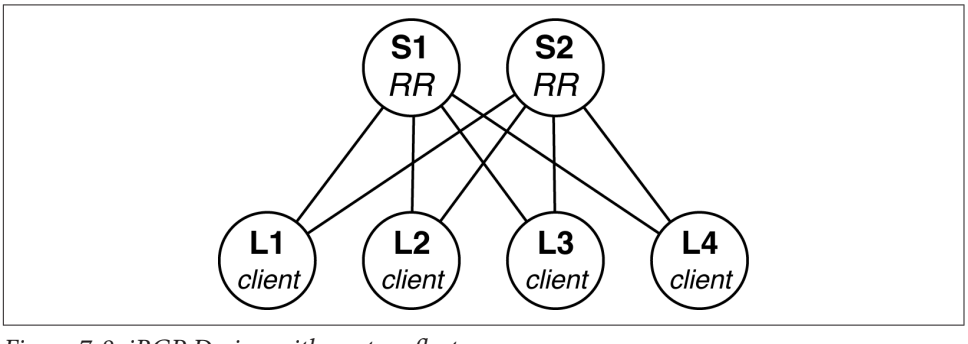


Figure 7-8. iBGP Design with route reflectors

It's important to ensure that the spine switches support BGP route reflection if you want to use an iBGP design. Fortunately, you're in luck, because the Juniper QFX5100 series supports BGP route reflection.

The other critical implementation requirement that you must meet in an iBGP design is ECMP routing. By default, BGP route reflectors only reflect the *best* route. This means that if four ECMP routes exist, only the *single, best* prefix is reflected to the clients. Obviously, this breaks the ECMP requirement and something must be done.

The answer to this problem is to enable the BGP route reflector to send multiple paths instead of the best. There is currently a draft in the IETF that implements this behavior. The feature is called BGP Add Path; with it, the route reflector can offer all ECMP routes to each client.

Ensure that the spine switches in your IP Fabric support BGP Add Path if you want to design the network using iBGP. Thankfully, the Juniper QFX5100 family supports BGP Add Path as well as BGP route reflection.

To summarize, the spine switches must support BGP route reflection as well as BGP Add Path to meet all of the IP Fabric requirements with iBGP, but iBGP does allow you to manage the entire IP Fabric as a single ASN.

eBGP Design

The other alternative is to use eBGP to design the IP Fabric. By default, eBGP will meet all of the implementation requirements when building the IP Fabric. There's no need for BGP route reflection or BGP Add Path with eBGP.

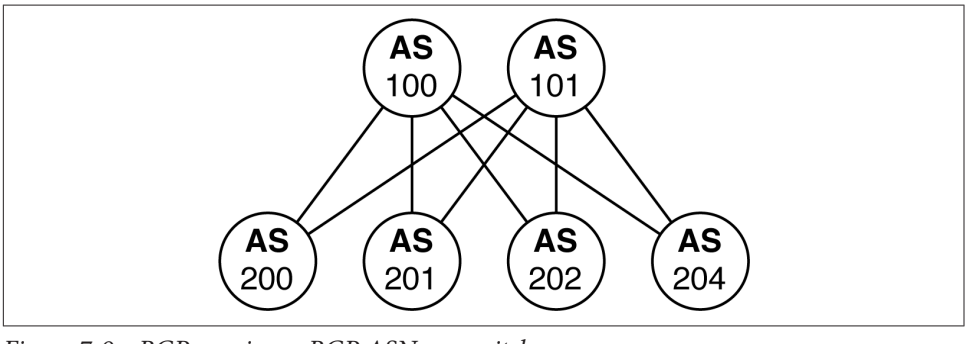


Figure 7-9. eBGP requires a BGP ASN per switch

The only thing you really have to worry about is how many BGP ASNs you will consume with the IP Fabric. Each switch will have its own BGP ASN. Technically the BGP private range is 64,512 to 65,535 (and 65,535 is reserved) which leaves you with 1,023 BGP ASNs. If the IP Fabric is larger than 1,023 switches, you're going to have to consider moving into the public BGP ASN range or move to 32-bit ASN numbers.

As you can see, eBGP has the simplest design that meets all of the implementation requirements. This works well when creating a multivendor IP Fabric.

Edge connectivity

The other critical decision point is how do you connect your data center to the rest of the world and to other data centers? There are multiple decision points due to the number of end points and options. Let's review a simple example of two data centers. Figure 7-10 gives you an overview.

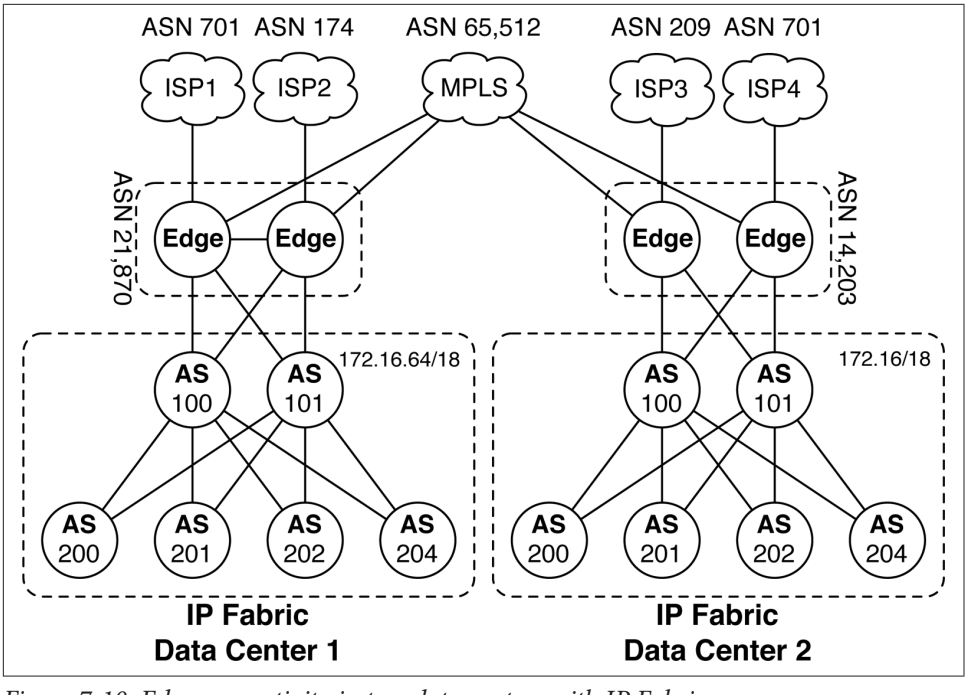


Figure 7-10. Edge connectivity in two data centers with IP Fabrics

Each data center has the following components:

- An IP Fabric using the same BGP ASN numbers and scheme
- Two edge routers with a unique BGP ASN
- They are connected to two ISPs
- They are connected to a private Multiprotocol Label Switching (MPLS) network

Reusing the same eBGP design in each data center reduces the operational burden of bringing up new data centers; it also creates a consistent operational experience, regardless of which data center you're in. The drawback is that using the same AS numbers throughout the entire design makes things confusing in the MPLS core. For example, what BGP prefixes does AS 200 own? The answer is that it depends on which data center you're in.

One simple solution is to use the BGP AS Override feature. This allows the PE routers in the MPLS network to change the AS used by the edge routers in each data center. Now, we can simply say that ASN 21,870 owns the aggregate 172.16.64/18 and ASN 14,203 owns the aggregate 172.16/18. For example, from the perspective of Data Center 1, the route to 172.16/18 is through BGP ASN 65,512 then 14,203. To do this,

you must create a BGP export policy on the edge routers in each data center that rejects all of the IP Fabric prefixes but instead advertises a single BGP aggregate.

When connecting out to the Internet, the design is a little different. The goal is that the IP Fabric should have a default route of 0/0, but the edge routers should have a full Internet table. Each data center has its own public IP range that needs to be advertised out to each ISP, as well. In summary, the edge routers will perform the following actions:

- Advertise a default route into the IP Fabric
- Advertise public IP ranges to each ISP
- Reject all other prefixes

IP Fabrics Review

With IP Fabrics, you can create some very large networks that are easily able to support overlay networking architectures in the data center. There are a few decision points that you must consider carefully when creating your own IP Fabric. How many switches will you deploy? Do you want to design for a multivendor environment using BGP features? How many data centers will be connected to each other? These are the questions you must ask yourself and consider into the overall design of each data center.

BGP Implementation

Let's get down to brass tacks. Moving from the design phase to the implementation phase requires physical devices, configurations, and verification. This section walks through the implementation in detail using Junos. In this laboratory we will have two spines and three leaves, as shown in Figure 7-11.

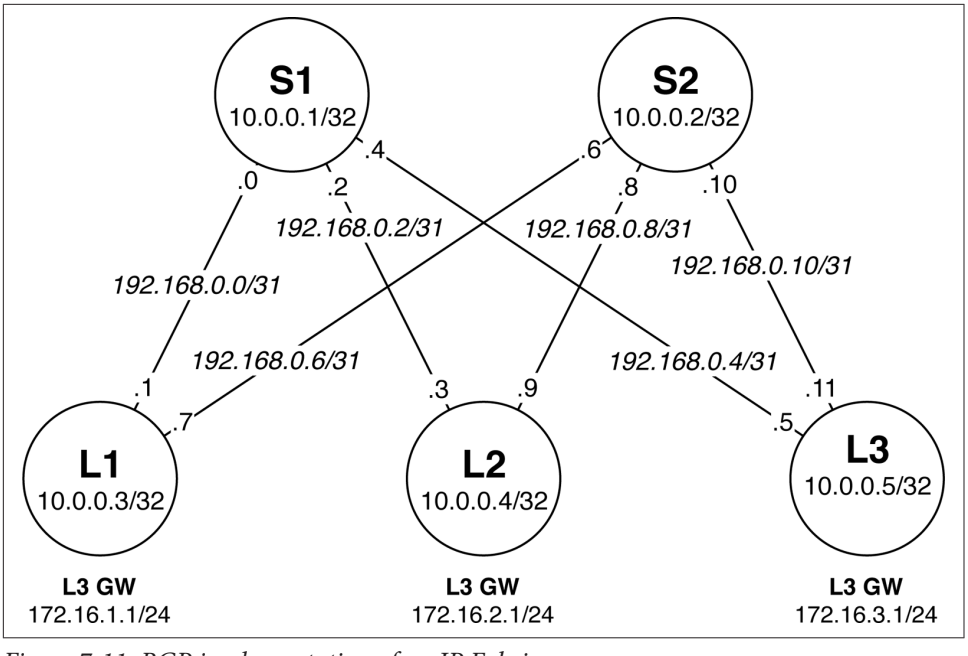


Figure 7-11. BGP implementation of an IP Fabric

There's a lot happening in Figure 7-11. The following is a breakdown of the IP address schema:

Loopback Address
: Each switch will use a 32-bit loopback address from the 10/8 range.

Point-to-Point Addresses
: Each switch will use a 31-bit netmask on each point-to-point link starting from 192.168/24.

Layer 3 Server Gateway
: Servers connecting to the IP Fabric will require a default gateway. Each leaf will have gateway services starting at 172.16.1/24.

Topology Configuration

The first step is to examine the topology and understand how each switch is connected, what the BGP attributes are, and IP address schemes. Each switch has a hostname, loopback, L3 gateway, and a BGP ASN. Table 7-2 lists the implementation details for you.

Table 7-2. BGP implementation details

Switch	Loopback	L3 gateway	BGP ASN
S1	10.0.0.1/32	None	100
S2	10.0.0.2/32	None	101
L1	10.0.0.3/32	172.16.1.1/24	200
L2	10.0.0.4/32	172.16.2.1/24	201
L3	10.0.0.5/32	172.16.3.1/24	202

My apologies for using a public BGP ASN in my lab. Please don't do this in real life.

Interface and IP Configuration

Now, let's investigate the physical connection details of each switch. Table 7.3 presents the interface names, point-to-point network, and IP addresses.

Table 7-3. Interface and IP implementation details

Source switch	Source interface	Source IP	Network	Destination switch	Destination interface	Destination IP
L1	xe-0/0/14	.1	192.168.0.0/31	S1	xe-0/0/14	.0
L1	xe-0/0/15	.7	192.168.0.6/31	S2	xe-0/0/15	.6
L2	xe-0/0/16	.3	192.168.0.2/31	S1	xe-0/0/16	.2
L2	xe-0/0/17	.8	192.168.0.8/31	S2	xe-0/0/17	.8
L3	xe-0/0/18	.11	192.168.0.10/31	S1	xe-0/0/18	.10
L3	xe-0/0/19	.1	192.168.0.0/31	S2	xe-0/0/19	.0

Each leaf is connected to each spine, but notice that the spines aren't connected to one another. In an IP Fabric, there's no requirement for the spines to be directly connected. Given any single-link failure scenario, all leaves will still have connectivity to one another. The other detail is that an IP Fabric is all Layer 3. Traditional Layer 2 networks require connections between the spines to ensure proper flooding and propagation of the broadcast domains. Yet another reason to favor Layer 3 IP Fabric: no need to interconnect the spines.

BGP Configuration

One of the first steps is to configure each spine to peer via eBGP to each leaf. One trick to speed up the BGP processing in Junos is to keep all the neighbors in a single

BGP group. We can certainly do this because the import and export policies are identical, but only the peer AS and neighbor IP vary from leaf to leaf. Here's the BGP configuration of S1:

```
protocols {
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.1 {
                peer-as 200;
            }
            neighbor 192.168.0.3 {
                peer-as 201;
            }
            neighbor 192.168.0.5 {
                peer-as 202;
            }
        }
    }
}
```

Each leaf has its own neighbor statement with the proper IP address. In addition, each neighbor has its own specific peer AS; this allows all of the leaves in the IP Fabric to be placed under a single BGP group called CLOS.

There are a few global BGP options you'll want to enable so that you don't have to specify them for each group and neighbor.

`log-updown`
: This enables tracking of all BGP session state. All groups and neighbors shall inherit this option. Now, we can keep track of the entire IP Fabric from the point of view of each switch.

Import and Export Policies
: A common import and export policy is used across the entire IP Fabric; it doesn't make a difference if it's a leaf or a spine. We'll review the policy statements in more detail later in the chapter.

`graceful-restart`
: Of course, you want the ability to make policy changes to BGP without having to tear down existing sessions. To enable this functionality, you can enable the `graceful-restart` feature in Junos.

Under the `CLOS` BGP group, we also enable some high-level features:

`type external`
: This enables eBGP for the entire BGP group. Given that the IP Fabric is based on an eBGP design, there's no need to repeat this information for each neighbor.

`mtu-discovery`
: We're running jumbo frames on the physical interfaces. Allowing BGP to discover the larger MTU will help in processing control plane updates.

BFD
: To ensure that you have fast convergence, you'll offload the forwarding detection to BFD. In this example, we're using a 350 ms interval with a 3x multiplier.

`multipath multiple-as`
: To allow for ECMP across a set of eBGP neighbors, you need to enable the `multipath multiple-as` option.

BGP Policy Configuration

The real trick is writing the BGP policy for importing and exporting the prefixes throughout the IP Fabric. It's actually straightforward. We can craft a common set of BGP policies to be used across both spines and leaves, which results in a simple copy-and-paste operation.

First up is the BGP export policy:

```
    policy-options {
        policy-statement bgp-clos-out {
            term loopback {
                from {
                    protocol direct;
                    route-filter 10.0.0.0/24 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
            term server-L3-gw {
                from {
                    protocol direct;
                    route-filter 172.16.0.0/12 orlonger;
                }
                then {
                    next-hop self;
                    accept;
                }
            }
        }
    }
```

There's a lot happening in this policy.

`term loopback`
: The first order of business is to identify the switch's loopback address and export it to all other BGP peers. You can do this by looking at the directly connected interfaces that match 10/24 or longer bitmask; this will quickly identify all loopback addresses across the entire IP Fabric. Personally, I keep `next-hop self` in the policy just in case there's a change to iBGP in the future; this way prefixes are still exchanged and next-hops are valid.

`term server-L3-gw`
: We already know that each leaf has Layer 3 gateway services for the servers connected to it; the range is 172.16/12. This will match all of the server gateway addresses on each leaf. Of course, you'll apply the `next-hop self`, as well. Obviously, this has no effect on the spines and will only work on the leaves; it's great being able to write a single policy for both switches.

Default
: Each BGP policy has a default term at the very end. It isn't configurable, but follows the default rules of eBGP: advertise all eBGP and iBGP prefixes to the neighbor; otherwise, deny all other prefixes. This simply means that other BGP prefixes in the routing table will be advertised to other peers. You can stop this behavior by installing an explicit reject action at the end, but in this case you want the IP Fabric to propagate all BGP prefixes to all leaves.

Here's the import policy configuration:

```
policy-options {
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/24 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/12 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
}
```

Again, there is a lot happening in the import policy. At a high level, you want to be very selective about what types of prefixes you accept into the routing and forwarding table of each switch. Let's walk through each term in detail.

`term loopbacks`
: Obviously, you want each switch to have reachability to every other switch in the IP Fabric via loopback addresses. You will explicitly match on the 10/8 and allow all loopback addresses into the routing and forwarding table.

`term server-L3-gw`
: The same goes for server Layer 3 gateway addresses; each leaf in the IP Fabric needs to know about all other gateway addresses. You'll explicitly match on 172.16/12 to allow this.

`term reject`
: At this point, you've had enough. Reject all other prefixes. The problem is that if you didn't have a reject statement at the end of the import policy, the routing and forwarding tables would be trashed by all of the point-to-point networks. There's no reason to have this information in each switch, because it's only relevant to the immediate neighbor of its respective switch.

You simply export and import loopbacks and Layer 3 server gateways and propagate all prefixes throughout the entire IP Fabric. The best part is that you can reuse the same set of policies throughout the entire IP Fabric, as well. Copy and paste.

ECMP Configuration

Recall that we used the `multipath multiple-as` configuration knob in the BGP section. That alone only installs ECMP prefixes into the Routing Information Base (RIB). To take full ECMP from the RIB and install it into the Forwarding Information Base (FIB), you need to create another policy that enables ECMP and install it into the FIB. Here's the policy:

```
routing-options {
    forwarding-table {
        export PFE-LB;
    }
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
}
```

The `PFE-LB` policy simply says that for any packet being forwarded by the switch, enable load balancing; this enables full ECMP in the FIB. However, the existence of the `PFE-LB` policy by itself is useless; it must be applied into the FIB directly. This is done under `routing-options forwarding-table` by referencing the `PFE-LB` policy.

BGP Verification

Now that you have configured the IP Fabric, the next step is to ensure that the control plane and data plane are functional. We can verify the IP Fabric through the use of show commands to check the state of the BGP sessions, what prefixes are being exchanged, and passing packets through the network.

BGP State

Let's kick things off by logging into S1 and checking the BGP sessions:

```
dhanks@S1> show bgp summary
Groups: 1 Peers: 3 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                       6          6          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn
State|#Active/Received/Accepted/Damped...
192.168.0.1             200      12380      12334       0       3 3d 21:11:35
2/2/2/0              0/0/0/0
192.168.0.3             201      12383      12333       0       2 3d 21:11:35
2/2/2/0              0/0/0/0
192.168.0.5             202      12379      12333       0       2 3d 21:11:35
2/2/2/0              0/0/0/0
```

All is well, and each BGP session to each leaf is connected and exchanging prefixes. You can see that each session has two active, received, and accepted prefixes; these are the loopback and Layer 3 gateway addresses. So far, everything is great.

Let's dig further down the rabbit hole. You need to verify, from a control plane perspective, ECMP, graceful restart, and BFD. Here it is:

```
dhanks@S1> show bgp neighbor 192.168.0.1
Peer: 192.168.0.1+60120 AS 200 Local: 192.168.0.0+179 AS 100
  Type: External    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: Cease
  Export: [ bgp-clos-out ] Import: [ bgp-clos-in ]
  Options: <Preference LogUpDown PeerAS Multipath Refresh>
  Options: <MtuDiscovery MultipathAs BfdEnabled>
  Holdtime: 90 Preference: 170
  Number of flaps: 3
  Last flap event: Stop
  Error: 'Cease' Sent: 1 Recv: 1
  Peer ID: 10.0.0.3        Local ID: 10.0.0.1          Active Holdtime: 90
  Keepalive Interval: 30         Group index: 1    Peer index: 0
  BFD: enabled, up
  Local Interface: xe-0/0/14.0
  NLRI for restart configured on peer: inet-unicast
  NLRI advertised by peer: inet-unicast
  NLRI for this session: inet-unicast
  Peer supports Refresh capability (2)
  Stale routes from peer are kept for: 300
  Peer does not support Restarter functionality
  NLRI that restart is negotiated for: inet-unicast
```

```
  NLRI of received end-of-rib markers: inet-unicast
  NLRI of all end-of-rib markers sent: inet-unicast
  Peer supports 4 byte AS extension (peer-as 200)
  Peer does not support Addpath
  Table inet.0 Bit: 10000
    RIB State: BGP restart is complete
    Send state: in sync
    Active prefixes:              2
    Received prefixes:            2
    Accepted prefixes:            2
    Suppressed due to damping:    0
    Advertised prefixes:          3
  Last traffic (seconds): Received 1    Sent 25   Checked 42
  Input messages:  Total 12381Updates 3Refreshes 0Octets 235340
  Output messages: Total 12334Updates 7Refreshes 0Octets 234634
  Output Queue[0]: 0
```

The important bits are italicized. Take a closer look at the two lines of `Options`. You can see the following:

- Logging the state of the BGP session
- Support ECMP
- Support graceful restart
- MTU discovery enabled
- BFD is bound to BGP

BGP Prefixes

With BGP itself configured correctly, let's examine what it's doing. Take a closer look at S1 and see what prefixes are being advertised to L1:

```
dhanks@S1> show route advertising-protocol bgp 192.168.0.1 extensive

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
* 10.0.0.1/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self
     Flags: Nexthop Change
     AS path: [100] I

* 10.0.0.4/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.3)
     AS path: [100] 201 I

* 10.0.0.5/32 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.5)
     AS path: [100] 202 I

* 172.16.2.0/24 (1 entry, 1 announced)
 BGP group CLOS type External
     Nexthop: Self (rib-out 192.168.0.3)
```

```
        AS path: [100] 201 I

    * 172.16.3.0/24 (1 entry, 1 announced)
     BGP group CLOS type External
         Nexthop: Self (rib-out 192.168.0.5)
         AS path: [100] 202 I
```

Things are really looking great. S1 is advertising five prefixes to L1. Here's a breakdown:

10.0.0.1/32
: This is the loopback address on S1 itself. You're advertising this prefix to L1.

10.0.0.4/32
: This is the loopback address for L2. You're simply passing this prefix on to L1. You can see the AS path is `[100] 201 I`, which means that the route origin was internal and you can simply follow the AS itself back to L2.

10.0.0.5/32
: Same goes for the loopback address of L3. Passing it on to L1.

172.16.2.0/24
: This is the Layer 3 gateway address for L2. Passing it on to L1.

172.16.3.0/24
: Same goes for the Layer 3 gateway address for L3. Passing it on to L1.

Here's what you're receiving from the other leaves:

```
dhanks@S1> show route receive-protocol bgp 192.168.0.1

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.3/32            192.168.0.1                             200 I
* 172.16.1.0/24          192.168.0.1                             200 I

dhanks@S1> show route receive-protocol bgp 192.168.0.3

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.4/32            192.168.0.3                             201 I
* 172.16.2.0/24          192.168.0.3                             201 I

dhanks@S1> show route receive-protocol bgp 192.168.0.5

inet.0: 53 destinations, 53 routes (52 active, 0 holddown, 1 hidden)
  Prefix  Nexthop      MED     Lclpref    AS path
* 10.0.0.5/32            192.168.0.5                             202 I
* 172.16.3.0/24          192.168.0.5                             202 I
```

Again, you can confirm that each leaf is only advertising its loopback and Layer 3 gateway address into the spine.

Routing Table

You have verified that the prefixes are being exchanged correctly between the switches, At this juncture, it's time to ensure that the RIB is being populated correctly. The easiest way to verify this is to log in to L1 and verify that we see ECMP to the loopback address of L3:

```
dhanks@L1> show route 172.16.3.1/24 exact

inet.0: 54 destinations, 58 routes (53 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

172.16.3.0/24      *[BGP/170] 3d 10:55:14, localpref 100, from 192.168.0.6
                      AS path: 101 202 I
                     > to 192.168.0.0 via xe-0/0/14.0
                       to 192.168.0.6 via xe-0/0/15.0
                     [BGP/170] 3d 10:55:14, localpref 100
                       AS path: 100 202 I
                     > to 192.168.0.0 via xe-0/0/14.0
```

What we see here is that there are two next-hops to L3 from L1. This is a result of a proper BGP configuration using the `multipath multiple-as` knob.

Forwarding Table

The next step is to ensure that the forwarding table is being programmed correctly by the RIB. You verify this the same way. Start on L1 and verify to L3:

```
dhanks@L1> show route forwarding-table destination 172.16.3.1
Routing table: default.inet
Internet:
Destination        Type RtRef Next hop           Type Index NhRef Netif
172.16.3.0/24      user     0                    ulst 131070     5
                              192.168.0.0        ucst  1702     5 xe-0/0/14.0
                              192.168.0.6        ucst  1691     5 xe-0/0/15.0
```

Of course, what you see here are two next-hops: one toward `S1` (`xe-0/0/14`), and the other toward `S2` (`xe-0/0/15`).

Ping

A simple way to verify the data plane connectivity is to log in to L1 and source a ping from its Layer 2 gateway address and ping L3; this will force traffic through the spine of the network:

```
dhanks@L1> ping source 172.16.1.1 172.16.3.1 count 5
PING 172.16.3.1 (172.16.3.1): 56 data bytes
64 bytes from 172.16.3.1: icmp_seq=0 ttl=63 time=3.009 ms
64 bytes from 172.16.3.1: icmp_seq=1 ttl=63 time=2.163 ms
64 bytes from 172.16.3.1: icmp_seq=2 ttl=63 time=2.243 ms
64 bytes from 172.16.3.1: icmp_seq=3 ttl=63 time=2.302 ms
64 bytes from 172.16.3.1: icmp_seq=4 ttl=63 time=1.723 ms

--- 172.16.3.1 ping statistics ---
```

```
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.723/2.288/3.009/0.414 ms
```

So far, so good. The trick here is to source the ping from the Layer 3 gateway address; that way you know that L3 has a return route for L1.

Traceroute

To get the next level of detail, let's use traceroute. You can verify that traffic is moving through the spine of the IP Fabric. Try mixing it up a bit by using the loopback addresses, instead:

```
dhanks@L1> traceroute source 10.0.0.3 10.0.0.5
traceroute to 10.0.0.5 (10.0.0.5) from 10.0.0.3, 30 hops max, 40 byte packets
 1  192.168.0.6 (192.168.0.6)  2.031 ms  1.932 ms 192.168.0.0 (192.168.0.0)
2.121 ms
 2  10.0.0.5 (10.0.0.5)  2.339 ms  2.342 ms  2.196 ms
```

What's interesting here is that we can see the traceroute happened to go through S2 to get to L3. This is just a result of how the traceroute traffic was hashed by the forwarding table of L1.

Configurations

If you would like to build your own IP Fabric and use this laboratory as a foundation, feel free to use the configuration. For the sake of page count, I share just a single spine and leaf switch. Being an astute reader, you can figure out the rest.

S1

Here is the configuration for S1:

```
interfaces {
    xe-0/0/14 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.0/31;
            }
        }
    }
    xe-0/0/16 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.2/31;
            }
        }
    }
    xe-0/0/18 {
        mtu 9216;
```

```
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.4/31;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.1/32;
            }
        }
    }
}
routing-options {
    router-id 10.0.0.1;
    autonomous-system 100;
    forwarding-table {
        export PFE-LB;
    }
}
protocols {
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.1 {
                peer-as 200;
            }
            neighbor 192.168.0.3 {
                peer-as 201;
            }
            neighbor 192.168.0.5 {
                peer-as 202;
            }
        }
    }
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/24 orlonger;
            }
```

```
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/12 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
    policy-statement bgp-clos-out {
        term loopback {
            from {
                protocol direct;
                route-filter 10.0.0.0/24 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term server-L3-gw {
            from {
                protocol direct;
                route-filter 172.16.0.0/12 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
    }
}
```

L1

And now, the L1 configuration:

```
interfaces {
    interface-range ALL-SERVER {
        member-range xe-0/0/0 to xe-0/0/13;
        member-range xe-0/0/16 to xe-0/0/47;
        unit 0 {
            family ethernet-switching {
                port-mode access;
                vlan {
                    members SERVER;
                }
            }
        }
    }
    xe-0/0/14 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.1/31;
```

```
            }
        }
    }
    xe-0/0/15 {
        mtu 9216;
        unit 0 {
            family inet {
                mtu 9000;
                address 192.168.0.7/31;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.3/32;
            }
        }
    }
    vlan {
        mtu 9216;
        unit 1 {
            family inet {
                mtu 9000;
                address 172.16.1.1/24;
            }
        }
    }
}
routing-options {
    router-id 10.0.0.3;
    autonomous-system 200;
    forwarding-table {
        export PFE-LB;
    }
}
protocols {
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.0 {
                peer-as 100;
            }
            neighbor 192.168.0.6 {
                peer-as 101;
            }
        }
    }
```

```
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/24 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/12 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
    policy-statement bgp-clos-out {
        term loopback {
            from {
                protocol direct;
                route-filter 10.0.0.0/24 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term server-L3-gw {
            from {
                protocol direct;
                route-filter 172.16.0.0/12 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term reject {
            then reject;
        }
    }
}
vlans {
    SERVER {
        vlan-id 1;
        l3-interface vlan.1;
    }
}
```

Summary

This chapter covered the basic ways to build an IP Fabric. More important, it has reviewed the decision points you must take into account when building an IP Fabric. There are various options in the control plane that impact what features are required on the platform. Finally, we reviewed in great detail how to implement BGP in an IP Fabric. We walked through all of the interfaces, IP addresses, BGP configurations, and policies. To wrap things up, we verified that BGP is working across the IP Fabric and ran some tests on the data plane to ensure that traffic can get from leaf to leaf.

Building an IP Fabric is a straightforward task and serves as a great foundation to overlay technologies such as VMware NSX and Juniper Contrail. Basing the design on only Layer 3 makes the IP Fabric very resilient to failure and offers very fast end-to-end convergence with BFD. Build your next IP Fabric with the Juniper QFX5100 series.

Chapter Review Questions

1. How do you avoid full mesh requirements in an iBGP design?

a. BGP route reflection

b. BGP confederations

c. All of the above

d. None of the above

2. What is required to enable ECMP in an iBGP design?

a. Multiple route reflectors

b. Enable load balancing

c. BGP Add Path

d. `multipath multiple-as`

3. How many private BGP ASNs are there?

a. 1,023

b. 1,024

c. 511

d. 512

Chapter Review Answers

1. ***Answer: C.*** *Both are valid options. I recommend using BGP route reflectors. The Juniper QFX5100 family can easily handle both.*

2. ***Answer: C.*** *You require BGP Add Path for the route reflector to advertise all equal next-hops instead of the best.*

3. ***Answer: A.*** *There are 1,023 private BGP ASNs.*

CHAPTER 8
Overlay Networking

What is it and what problems can we solve with it?

These are the first questions that any good network engineer will ask when he first encounters a new technology or feature.

One of the larger problems in the data center is being able to easily orchestrate compute, storage, and networking to provide data center services with a click of a mouse. There are tools such as OpenStack, CloudStack, VMware vSphere and others that can help you accomplish this goal. The problem with these tools is that special plugins are required to help orchestrate the network. For example, if a new virtual machine (VM) was created and required to be on a separate network, the orchestration software would have to use a plugin to automatically configure the network switches with new Virtual Local Area Networks (VLANs), default gateways, and Access Control Lists (ACLs). The problem is that plugins only offer basic functionality and the feature parity varies between vendors. The other problem is that every time you create a new VM, add a NIC to an existing VM, or move a VM, it requires a change to the physical network.

One method of solving these problems is to decouple the virtual network from the physical network. The basic premise being that if all changes were made on the virtual networks, the physical network wouldn't require changes. The tradeoff is that you're moving complexity from one location to another. In this example, we're moving physical network changes to a virtual concept. However, the benefit is that we can now abstract the way virtual networks are created, which reduces the operational complexity of the physical network. For example if you built a physical network with multiple vendors, the assumption is that no changes are required on the physical network when there's a change to the virtual network. The benefit being that you don't need to orchestrate network changes across a set of physical switches by different vendors; you simply worry about orchestrating change across the abstracted virtual

network. Doing something the same way every time is much easier than doing the same thing with different styles. In a nutshell, overlay networking in the data center creates a hardware abstraction layer for the physical network and provides programmatic consistency.

Overlay networking might sound very similar to how physical servers were virtualized and abstracted from their underlying hardware. The basic premise is exactly the same. Virtualize the physical server and create a new virtual server that's agnostic with respect to the underlying hardware. No need to worry about installing different operating system drivers to support different storage or networking cards. From the perspective of the VM, all of the operating system drivers are standardized, regardless if the underlying physical server uses solid-state drives (SSDs) or remotely mounted iSCSI targets. The same holds true for overlay networking in the data center. When you create a virtual network, it doesn't have to worry about underlying protocols such as Spanning Tree Protocol (STP), Multi-Chassis Link Aggregation (MC-LAG), or Open Shortest Path First (OSPF).

As of this writing, overlay networking in the data center has started to trend and gain some adoption. The two primary options for data center overlay solutions are Juniper Contrail and VMware NSX. Each of these solutions creates virtual networks and decouples them from the physical hardware. You can orchestrate the virtual networks by using solutions such as OpenStack so that servers, storage, and networking are managed by a single tool.

Overview

As I mentioned in the introduction to this chapter, one of the key questions you should always ask when exploring a new piece of technology is what problem can you solve with it that you're unable to solve today? If the answer is nothing new, the immediate follow-up question you should ask is does the new technology have a tangible benefit when compared with existing technology? The answer can come in multiple forms. For example, new technology might perform faster, have higher scale, or execute more quickly. If the new technology doesn't offer any tangible benefits when compared with older technology, the industry refers to this as a "solution looking for a problem."

So, what problem does overlay networking in the data center solve that can't be solved today? The simple answer is that overlay networking gives you the capability to programmatically create logical networks in a standardized workflow without having to worry about the underlying hardware. Do you have a Juniper network? No problem, you can spin up logical networks using a standard Application Programming Interface (API). Do you have a Juniper network with other networking gear from other vendors? No problem. Use the same API to create logical networks.

There's also another problem that exists in the data center. Standard access switches that are used as top-of-rack (ToR) switches are made by using merchant silicon (also referred to as "off-the-shelf silicon") such as the Broadcom Trident 2 chipset. These switches have a limited Ternary Content Addressable Memory (TCAM) and can only scale so far in terms of MAC addresses, host entries, and IP address prefixes. Traditionally if you needed larger scale, you had to move away from merchant silicon and on to something that's purpose built for high scale such as Juniper silicon. Two good examples of Juniper silicon are the Juniper MX routers and Juniper EX9200 switches; these products have much higher scale than their counterparts in the access switches. The problem is that you can't use a Juniper MX or Juniper EX9200 as an access switch. Overlay networking moves the scale outside of the network and into the servers. All MAC addresses, host entries, and IP address prefixes are now stored in standard servers using x86 CPUs and large amounts of memory, which offer much greater scale than merchant silicon found in access switches.

Clearly, there are some unique benefits to overlay networking in the data center: standardized programmatic creation of logical networks and much higher scale are but two. How can you use these new advantages in a real-world use case? Two of the most popular use cases that benefit from overlay networking are IT-as-a-Service and Infrastructure-as-a-Service. Let's take a look at each of these in more detail.

IT-as-a-Service

One of the most popular use cases for overlay networking is IT-as-a-Service (ITaaS), which you can see in Figure 8-1. The scenario is that a business would implement ITaaS so that internal corporate users could simply request IT resources through a self-service portal. The user would be presented with a menu of services such as the following:

- Small VM: 1 core, 512 MB of memory, and 2 GB of storage
- Medium VM: 2 cores, 2 GB of memory, and 10 GB of storage
- Large VM: 4 cores, 8 GB of memory, and 50 GB of storage

The end user would select which item she wants from the menu of services and submit it into the self-service portal, as shown in the first step in Figure 8-1. The next step is that the self-service portal automatically creates the VM in the data center. The final step is that the VM is online and available to the user.

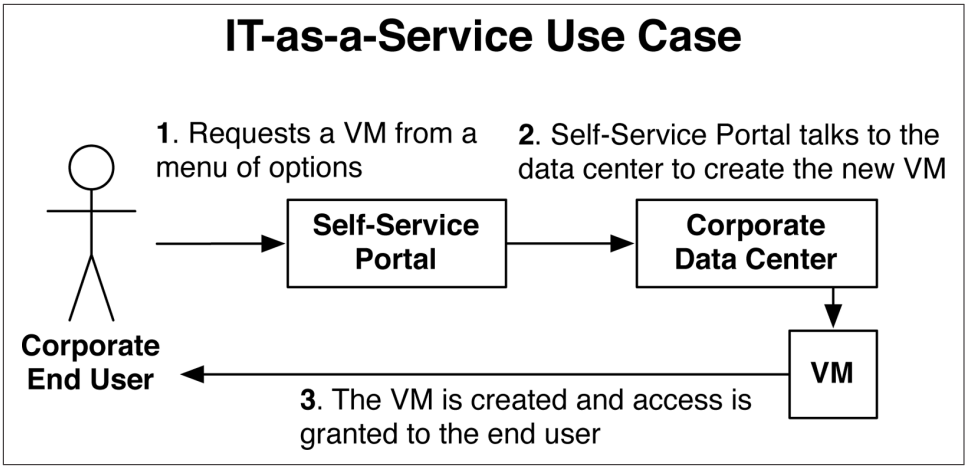


Figure 8-1. An ITaaS use case

The benefit is that end users can create, modify, and delete VMs without using a ticketing system that traditionally requires the work of three other people: a server administrator, a storage administrator, and a network engineer. When people are involved in the process, the time to create and deliver a VM back to the end user could take weeks. With ITaaS, the entire workflow is automated and the VM is delivered to the end user within minutes.

Infrastructure-as-a-Service

The second most common use case for overlay networking is Infrastructure-as-a-Service (IaaS). The scenario is that a service provider already has an existing Multi-Protocol Label Switching (MPLS) network from which customers can buy transport services. The customers need a large WAN to interconnect their sites (see Figure 8-2). For example, the headquarters of Customer-1 is connected to his retail stores; and the headquarters of Customer-1 is connected to all of his branch offices. So far, this is a standard service provider offering around MPLS. The IaaS comes into play when the service provider offers managed VMs to customers over their existing MPLS transport. For example, Customer-3 can buy a VM from the service provider's hosted data center and be able to access his private VM from his headquarters and data center. The same is true for Customer-1; she can buy a private VM from the service provider and access the VM directly from her retail stores through the service provider's MPLS network, as illustrated in Figure 8-2.

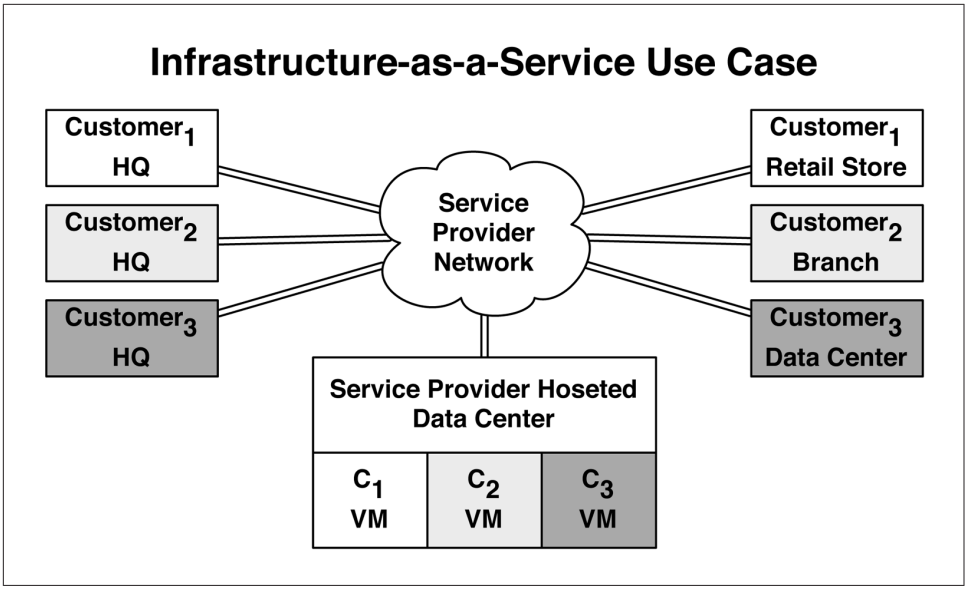


Figure 8-2. An IaaS use case

The benefit to the customers is that they can simply buy private VMs from the service provider and not have to worry about the underlying infrastructure. To make things even better, the customers can use their existing WAN connections from the service provider to access the VMs. Any customer locations such as headquarters, branch offices, or data centers that are connected into the service provider network will have connectivity to the VMs. The benefit to the service provider is that it can reuse its existing MPLS network to deliver new services to an existing customer base.

The Rise of IP Fabrics

There's a fundamental shift in the way data centers are built when moving to an overlay network. Because the server traffic is encapsulated and transmitted through overlay tunnels, the networking requirements have been reduced to support only Layer 3. Even when two VMs require Layer 2 connectivity, you can do this through an overlay network between the two VMs; the only core requirement on the underlying network is to support Layer 3.

Building a data center network on top of a routed, Layer 3 network is inherently more stable because it only has to support a single routing protocol. The requirement for Layer 2 has been completely removed and you no longer need to worry about STP, MC-LAG, and other Layer 2 protocols. A pure Layer 3 network is also able to scale higher in terms of physical devices and logical routing. Each switch in the network can simply run a routing protocol with every other switch in the network and take advantage of full Equal-Cost Multipath (ECMP). Because each switch is in full Layer

3 mode, it doesn't need to worry about propagating MAC addresses and can adjust its TCAM to support more Layer 3 entries.

Taking advantage of my personal experience in the data center, I've found that there's an even split between traditional networks and Ethernet Fabrics in the enterprise market through tier-1 service providers (see Figure 8-3), with traditional networks being defined as a core switch running Layer 3, distribution switches running both Layer 2 and Layer 3, and the access switches in Layer 2 mode, and Ethernet Fabrics being defined as a solution such as Juniper QFabric and Virtual Chassis Fabric. As of this writing, only a small number of customers are moving toward IP Fabrics and overlay networks.

However, there is a segment of customers in the web-services market called Massively Scalable Data Centers (MSDC) that fully use IP Fabrics. Most of these customers do not virtualize their workloads and have no requirement for overlay networking. Instead, they have custom-written applications, built-in redundancy, and no requirement for Layer 2. The benefit being that large web-scale companies can build data centers that house 100,000s of servers and the application availability is very high.

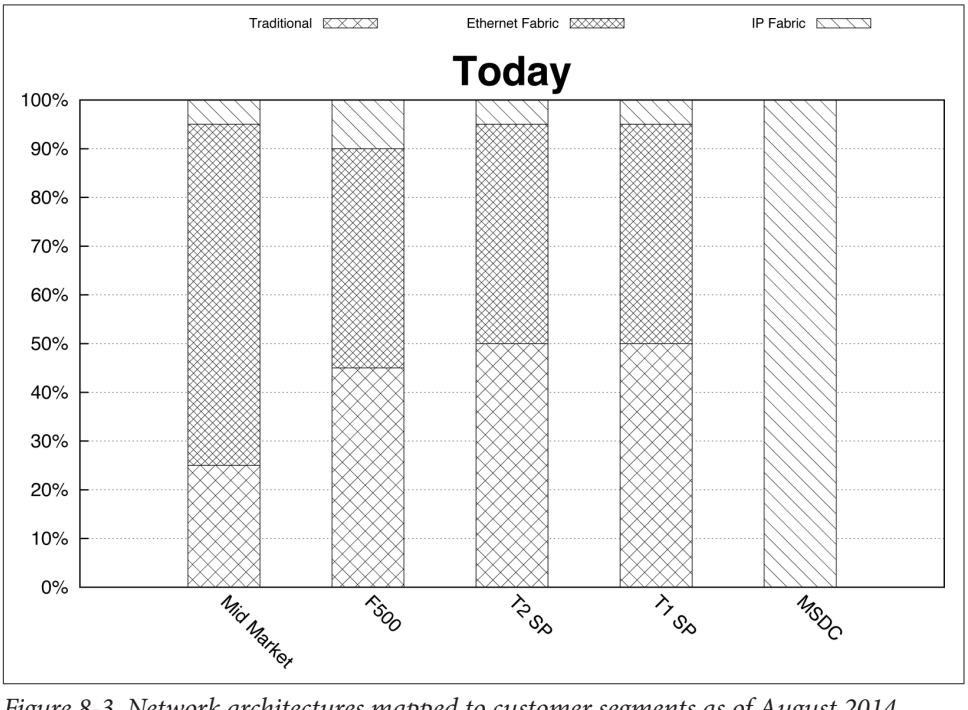


Figure 8-3. Network architectures mapped to customer segments as of August 2014

Given the benefits of overlay networking in the enterprise, it's fully expected that as market adoption takes place, a larger percentage of enterprise customers will move toward an IP Fabric network architecture as depicted in Figure 8-4.

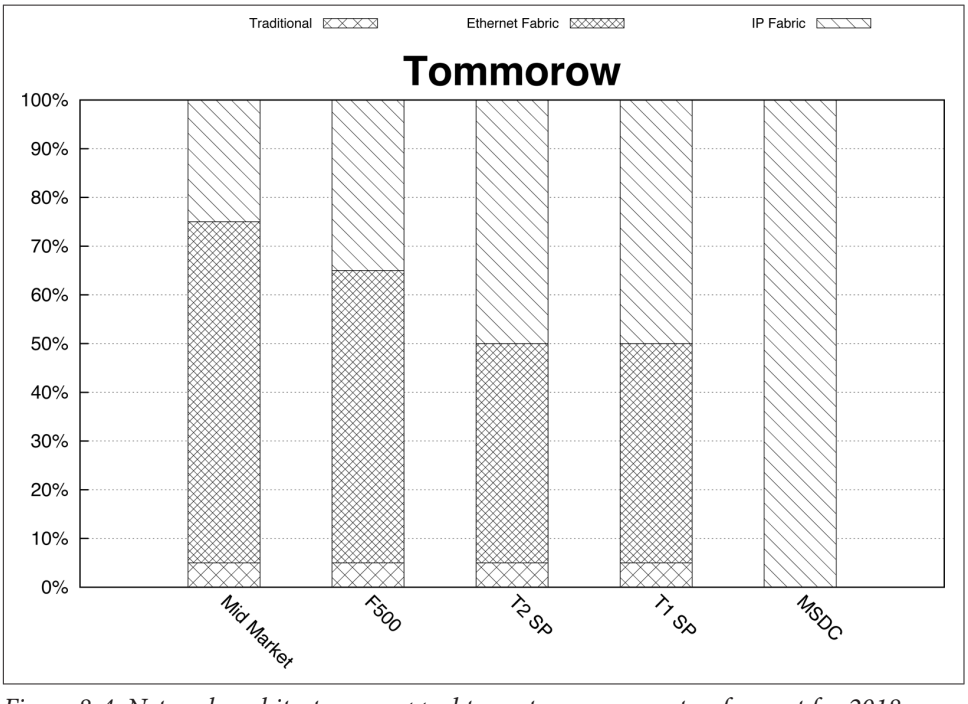


Figure 8-4. Network architectures mapped to customer segments—forecast for 2018

Today, there is a large percentage of enterprise and service-provider customers that are moving from a traditional network architecture to an Ethernet Fabric architecture, regardless of where overlay networking is going. The driving factors to use an Ethernet Fabric architecture is reduced operational complexity, storage convergence, and full support for Layer 2 and Layer 3 ECMP without depending on STP and MC-LAG.

To sum up, given the existing migration from traditional architectures to Ethernet Fabric architectures, plus the benefits of overlay architectures, it's my assertion that within five years, there will be an even split between Ethernet Fabrics and IP Fabrics. There will be no change in the way the MSDC customers do business; it's predicted that they will stay with an IP Fabric architecture and continue providing web-scale Software-as-a-Service (SaaS) applications.

Architecture

Overlay networking in the data center has many moving parts that are required to make it all work. The promise of being able to programmatically create virtual networks with standard APIs and not having to worry about the physical network isn't an easy task. There are layers of abstraction that must work together to create a true end-to-end service offering to the network operator and end users.

The first thing to understand about overlay networking is that it gets its name from building overlay networks between the hypervisor hosts (see Figure 8-5). By creating overlay tunnels between hypervisor hosts, the underlying network simply needs to handle Layer 3 connectivity. Figure 8-5 shows the entire data center network as a Layer 3 cloud. When VMs need to communicate with one another, the VM-to-VM traffic is encapsulated into an overlay tunnel and transmitted across the underlying Layer 3 network infrastructure until it reaches the destination hypervisor. The destination hypervisor receives the VM-to-VM traffic through the overlay tunnel, and it will decapsulate the traffic and forward it to the destination VM.

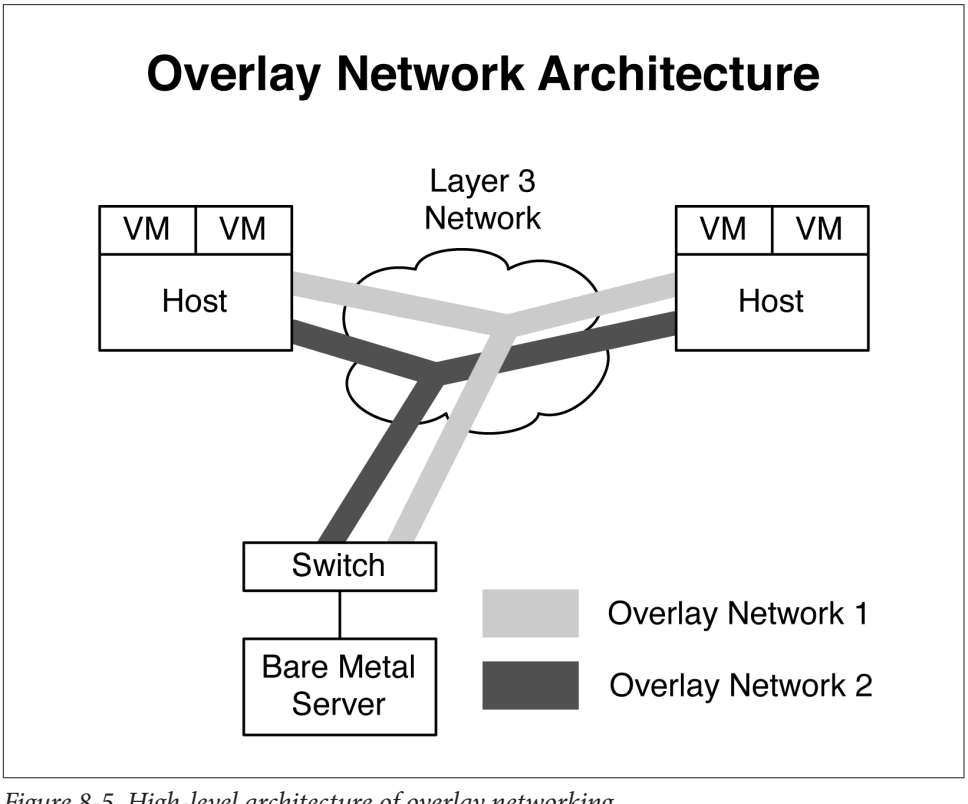


Figure 8-5. High-level architecture of overlay networking

Overlay networking works well when there's nothing but VMs in your data center. The tricky part is how to handle nonvirtualized servers such as bare-metal servers and appliances. These servers expect to speak native Ethernet to the switch and do not have the benefit of a hypervisor handling the overlay tunnels on their behalf. The answer is that the access switch itself can handle the overlay tunnel encapsulation and forwarding on behalf of the nonvirtualized servers, as demonstrated in Figure 8-5. From the perspective of a bare-metal server, it simply speaks standard Ethernet to the access switch, and then the access switch handles the overlay tunnels. The end result is that both virtual and nonvirtualized servers can use the same overlay architecture in the data center.

Controller-Based Overlay Architecture

One of the most common types of overlay architectures includes the use of a controller. The basic premise being that all of the Virtual Tunnel End Points (VTEPs) are provisioned and MAC address learning is handled through a centralized controller. The benefit to a controller-based architecture is that the MAC address learning happens in the control plane, as opposed to the data plane, which is less efficient. A controller-based architecture also provides a single point of management for the provisioning of virtualized networks through the creation of tunnels and VTEPs. Let's put all of the pieces together in a more detailed view of the end-to-end architecture for overlay networking. Figure 8-6 shows how virtualized and nonvirtualized servers are able to build overlay tunnels between the hypervisors and leaf switches.

All of the overlay tunnel end points are terminated in the host hypervisor or in the leaf switch. Notice that the overlay controller is connected to each tunnel end point. The controller handles the creation of overlay tunnels and MAC address learning. The overlay controller is connected to an overlay manager, which has a set of APIs. Cloud management software (CMS) such as OpenStack can use these APIs for network orchestration.

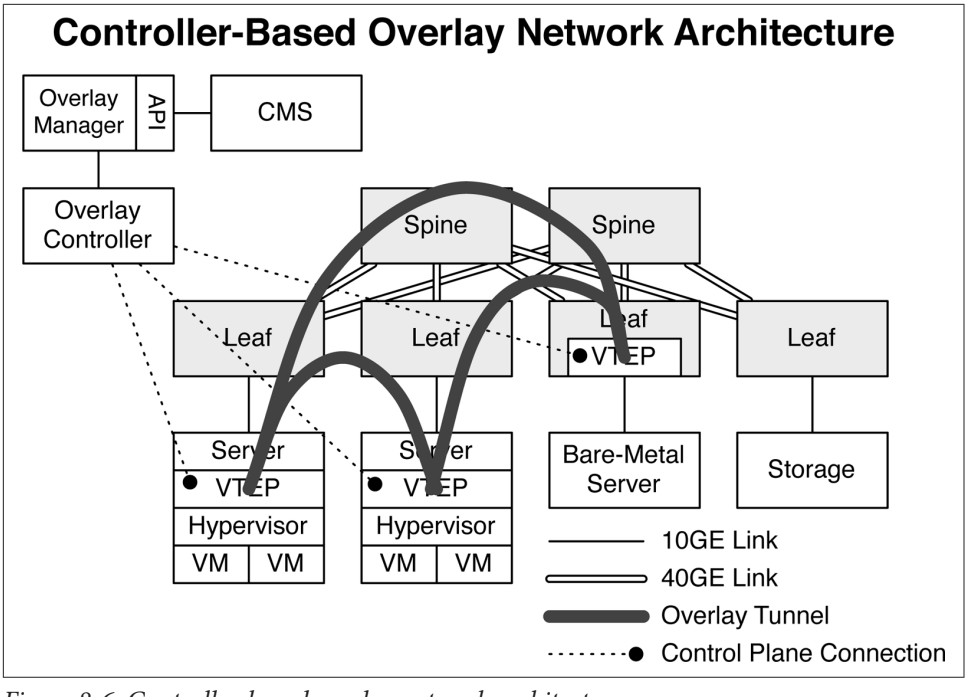


Figure 8-6. Controller-based overlay network architecture

Controller-Less Overlay Architecture

The alternative to a controller-based overlay architecture is a controller-less overlay architecture. The concept is that there is no centralized controller that's handling the MAC address learning.

Multicast

In the original Virtual Extensible LAN (VXLAN) draft, it was suggested to use multicast between VTEPs to enable MAC address learning. Multicast would simulate an Ethernet switch and simply forward all broadcast and unknown unicast traffic to every other VTEP in the network. The end result is that MAC addresses are flooded to every port in the IP Fabric and MAC address learning happens in the data plane, as shown in Figure 8-7.

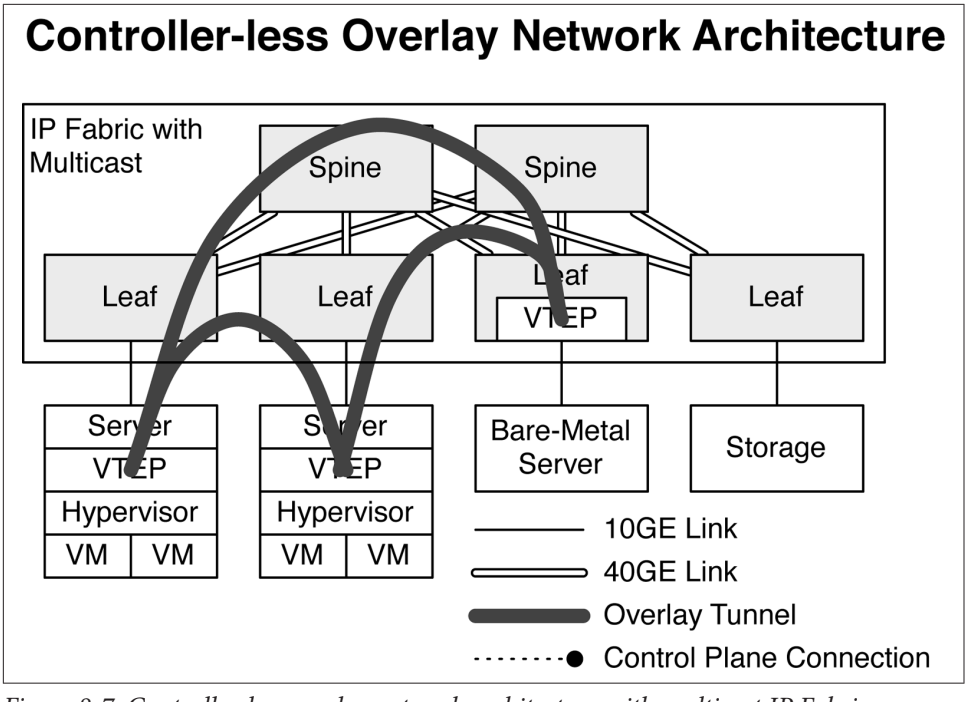


Figure 8-7. Controller-less overlay network architecture with multicast IP Fabric

The multicast option portrayed in Figure 8-7 is inefficient because it uses the data plane as a flooding mechanism for MAC address learning. It's very common to have multiple VTEPs in a data center, but each VTEP will have different VXLAN memberships. For example a VTEP on Leaf-1 may have a VXLAN membership of VNI-1, VNI-2, and VNI-3; VTEP on Leaf-2 may have a VXLAN membership of VNI-2, VNI-3, and VNI-4. Using multicast in this scenario would send broadcast and unknown cast traffic for VNI-1 to Leaf-2, although Leaf-2 doesn't have VNI-1 in the VTEP. The only way to work around this is to map VTEPs to a multicast group. However, this doesn't scale very well, because sooner or later you run out of multicast groups. Besides, who wants to maintain a VTEP to multicast group mapping and have to continually update it?

EVPN

The alternative to using multicast in a controller-less overlay architecture is to use a control plane protocol such as EVPN. Most people think that Ethernet VPN (EVPN) is tied directly to MPLS VPNs, but Juniper has decoupled the EVPN control plane from the data plane. You can now couple the EVPN control plane protocol with any data plane encapsulation, such as VXLAN, as shown in Figure 8-8.

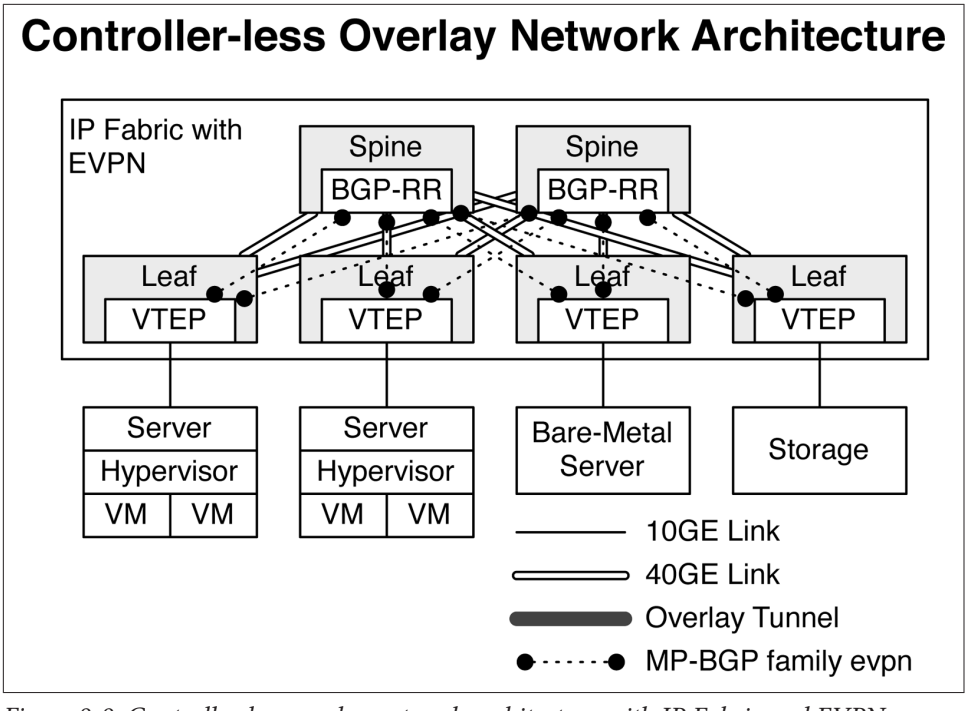


Figure 8-8. Controller-less overlay network architecture with IP Fabric and EVPN

EVPN is a very efficient control plane protocol because it uses standard Multiprotocol-Border Gateway Protocol (MP-BGP) extended communities such as route targets (RT) and route distinguishers (RD) to uniquely identify VTEP and VXLAN memberships. Now, VTEPs can exchange MAC addresses through EVPN with other VTEPs who are interested in specific VXLAN Network Identifiers (VNIs). The result is that MAC address learning is perfectly efficient when compared to multicast.

The controller-less overlay network architecture with EVPN is commonly referred to as a VXLAN Fabric. One of the other architectural changes in a VXLAN Fabric is that all of the VTEPs are now located in the networking equipment. The servers no longer need to worry about VTEPs and MAC address learning. All of the VXLAN switching, routing, and MAC address learning is handled by the networking equipment. From the perspective of the server, it's simply Ethernet.

The inter-VXLAN routing is handled in the spine with hardware that uses Juniper silicon, such as the Trio chipset. As of this writing, there is no available merchant silicon from Broadcom that's able to route VXLAN traffic. The default gateway of the servers is also handled by the spine switches, as illustrated in Figure 8-9.

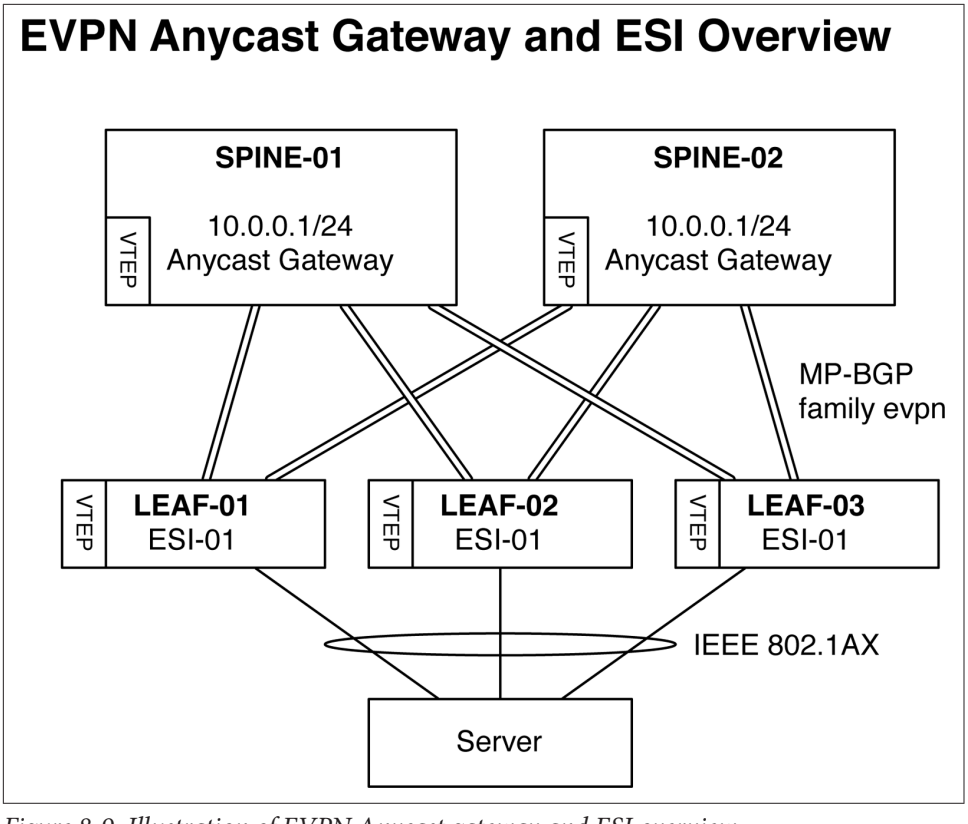


Figure 8-9. Illustration of EVPN Anycast gateway and ESI overview

The default gateway is synchronized with the EVPN protocol; each spine switch shares a common Anycast default gateway that's able to route traffic from any server. For example, in Figure 8-9, the server's default gateway would be 10.0.0.1. Also notice that the server is connected via LACP/IEEE 802.1AX to three switches: LEAF-01, LEAF-02, and LEAF-03. Depending on how the traffic is locally hashed within the server, the traffic can end up going to any of the three leaf switches. Because each of the three leaf switches share a common Ethernet Segment ID (ESI) for the server, each switch knows that any traffic that arrives from the server is part of the same virtual IEEE 802.1AX bundle.

EVPN has major benefits in a VXLAN Fabric architecture. The first is the reduction of wasted IP space for default gateways. Now, you can use the same IP address across a set of spine switches and serve as the Anycast default gateway for a particular broadcast domain. The second benefit is that EVPN allows a server to be multihomed into the network, as opposed to dual-homed. The advantage is that EVPN can span more than two leaf switches to provide the ultimate link resiliency.

Traffic Profiles

Let's take a look at the basic traffic patterns in an overlay network. It's important to understand where data can originate and where it can be destined. Depending on what the source and destination end points are, different functional roles in the architecture are used to ensure that traffic is delivered end to end. Figure 8-7 shows that in an overlay architecture, there are three basic traffic patterns that can take place:

- VM-to-VM traffic
- VM-to–physical server traffic
- Physical server–to–physical server traffic

Regardless of the traffic profile, the thing in common is that all traffic must pass through a VTEP. The VTEP is what terminates the tunnel at the source and destination of each overlay tunnel, as shown in Figure 8-10.

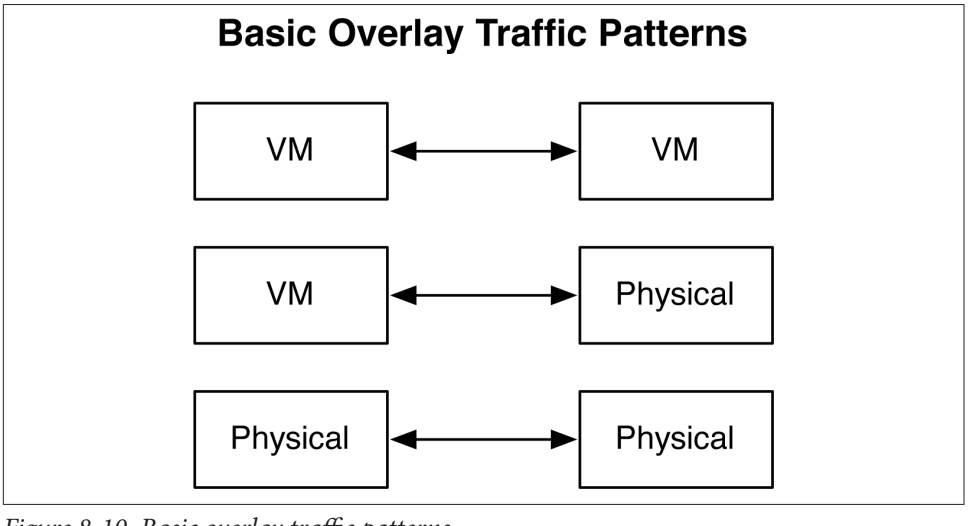


Figure 8-10. Basic overlay traffic patterns

VM-to-VM traffic will be handled by the VTEPs in the host hypervisor. The exception is that if a VM needs to communicate with a physical server, the traffic will still flow from VTEP to VTEP, but in this example, the host hypervisor VTEP will communicate with the access switch VTEP to handle the VM-to–physical server traffic. The final traffic profile is physical server–to–physical server. In this example, the traffic will be handled completely by access switch VTEPs at the source and at the destination. The end result is that regardless of the traffic profiles, overlay networking provides a seamless architecture to the network operator.

VTEPs

We already know that VTEPs terminate each end of an overlay tunnel. Let's make a closer examination to see how they actually work. There are three primary functions for each VTEP, as shown in Figure 8-11.

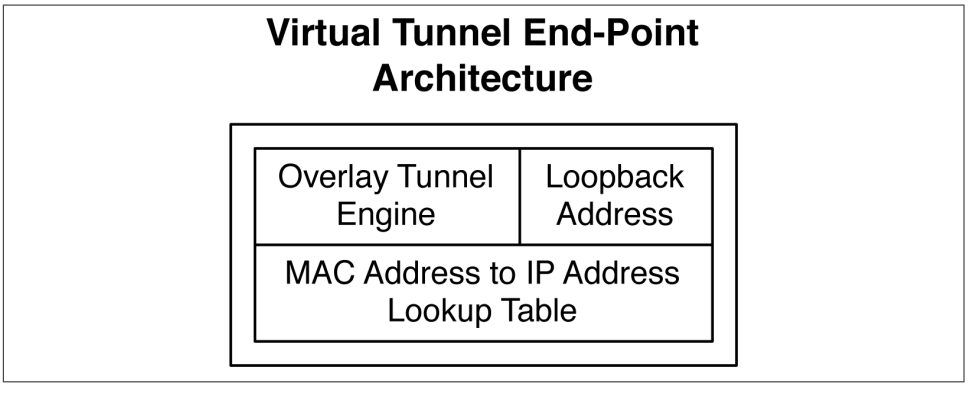


Figure 8-11. VTEP architecture

Overlay Tunnel Engine
: As VMs and physical servers transmit and receive traffic, that traffic needs to be placed into and out of the overlay tunnel. As traffic enters the overlay tunnel it will be encapsulated with the data plane of choice, which is typically VXLAN. When the traffic exits the other end of the tunnel, the overlay data plane needs to be removed and forwarded to the destination server.

Loopback Address
: As traffic is transmitted across an overlay tunnel, it requires a source and destination VTEP address. The loopback address provides the VTEP with Layer 3 reachability to other VTEPs in the network. Because VTEPs only require Layer 3 reachability between each other, this removes the underlying requirement for Layer 2 in the physical network.

MAC Address–to–IP Address Lookup Table
: One of the most critical roles of the VTEP is its ability to learn MAC addresses and map them to IP addresses and VTEP addresses. If MAC-1 was learned by VTEP-1, how would VTEP-2 know to forward any traffic with a destination address of MAC-1 to VTEP-1? The answer is that each VTEP has a global table of every MAC address in the network and which VTEP and IP address with which it's associated.

MAC address learning takes place inside the hypervisor or access switch and is replicated within each local VTEP. For every MAC address learned, its associated IP

address and local VTEP loopback address is populated into a VTEP table, as illustrated in Figure 8-12.

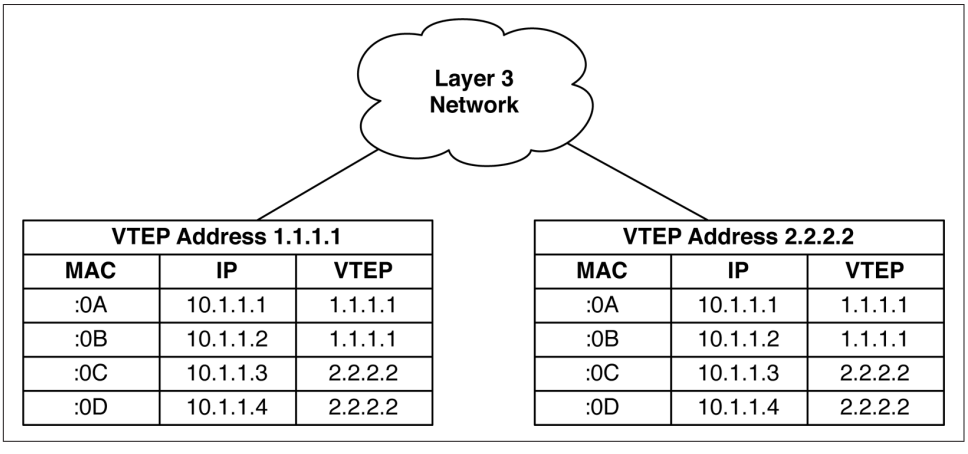


Figure 8-12. VTEP MAC, IP, and loopback tables

Each VTEP knows all local and remote MAC addresses and IP addresses. For example, if VTEP 1.1.1.1 wanted to send traffic to a destination VM with a MAC address ending in :0A, it knows that the VM is local and can simply perform local forwarding. However if VTEP 1.1.1.1 wanted to send traffic to a destination VM with the MAC address ending in :0D, it knows that the VM is owned by the VTEP with the address of 2.2.2.2. In this case, VTEP 1.1.1.1 would encapsulate the traffic destined to the MAC address ending in :0D in an overlay tunnel that was destined to the VTEP address 2.2.2.2 and transmit it over Layer 3. The destination VTEP 2.2.2.2 receives the encapsulated traffic and removes the overlay encapsulation. It inspects the destination MAC address and sees that :0D is a local MAC address and simply forwards the traffic to the VM. The same function is performed for both VMs and physical servers attached to access switches that have VTEPs.

Broadcom Trident II VTEPs

One caveat regarding the Broadcom Trident II VTEP is that it only maps MAC addresses to interfaces. This is because it's unable to route VXLAN traffic; it can only forward it across Layer 2. The end result is that something besides a Broadcom Trident II VTEP is required to route VXLAN traffic. Here are three alternatives:

Juniper Silicon
: Juniper switches such as the EX9200, and routers such as the MX series use custom-built Juniper silicon called the Trio chipset. These platforms make it possible for you to bind the MAC address, interface, and IP address within the chipset. The end result is that you can both switch and route VXLAN traffic.

Hypervisor VTEP
: VTEPs that reside within a hypervisor such as Linux KVM have a specialized function called a virtual router (vRouter). These vRouters are able to map the MAC addresses, interfaces, and IP addresses into tables in the hypervisor, which allow it to switch and route VXLAN traffic.

Custom Appliances
: The same principle applies, except the VXLAN routing is handled by a specialized piece of hardware in an appliance form factor. This option isn't very popular, but does exist.

Control Plane

Depending on the overlay solution used, there are various types of control plane options. At a high level, there are two methods to take care of MAC address learning between VTEPs: multicast and unicast. Multicast simply uses the data plane to replicate traffic between VTEPs; it's wildly inefficient, but it works. A more elegant method is to use a real control plane protocol with unicast traffic.

There are two major options when it comes to the control plane in an overlay network. The first option is the Open vSwitch Database (OVSDB), which is used by VMware NSX. The second option is the more well-known EVPN protocol, which is used by Juniper Contrail.

Juniper has led the industry in decoupling the EVPN control plane protocol from the underlying data plane encapsulations. The first appearance of EVPN was in the WAN with MPLS. Customers wanting to upgrade from VPLS to enable control plane MAC address learning and active-active ECMP use EVPN.

Juniper Contrail also uses EVPN, but with a MPLS over UDP data plane encapsulation. The functions are identical; use a control plane–based protocol for MAC address learning.

Solutions that take advantage of a unicast control plane require the use of an overlay controller. The overlay controller is the centralized mechanism for MAC address learning. As different VTEPs throughout the network learn MAC addresses, they update the overlay controller, as shown in Figure 8-6. In return, as the overlay controller learns new MAC addresses from other VTEPs, it replicates those MAC addresses to all other VTEPs in the network. The replication of MAC addresses throughout the network using a control plane protocol is very efficient and scales nicely.

Data Plane

As traffic moves between VMs and physical servers in an overlay network, it must be transmitted by an overlay encapsulation. Table 8-1 shows the various options that are available when it comes to the data plane encapsulation in an overlay architecture.

Table 8-1. Overlay architecture data plane encapsulation mapping

Product	VM-to-VM traffic	VM-to-physical traffic
Juniper Contrail	Agnostic VXLAN or GRE.	Agnostic VXLAN or GRE.
VMware NSX-MH	STT	VXLAN
VMware NSX-V	VXLAN	VXLAN

Juniper Contrail has kept an open mind in terms of what data plane encapsulation it uses. Because it has standardized on EVPN, it's able to use any type of data plane. VMware NSX has two products for overlay architectures:

VMware NSX for Multi-Hypervisor (NSX-MH)
: NSX-MH is focused for service providers and large enterprise customers that have a mixed environment of hypervisors such as Linux KVM and Xen. It has good support for OpenStack for data center orchestration. It also supports hardware accelerated VTEPs in networking switches using the OVSDB control plane protocol.

VMware NSX for vSphere (NSX-V)
: NSX-V is purely focused on enterprise customers who want a vertically integrated solution from VMware. For example, NSX-V has tight integration with VMware vSphere, vCloud Director, and VMware vCenter Operations. However, it doesn't support hardware-accelerated VTEPs in networking switches using the OVSDB control plane protocol. If you need to handle physical servers in the network, it must be performed in software through a VMware Service VM.

VXLAN

The most popular data plane encapsulation for overlay tunnels is VXLAN. It adds an additional 64 bits to the overall packet size, as shown in Figure 8-13.

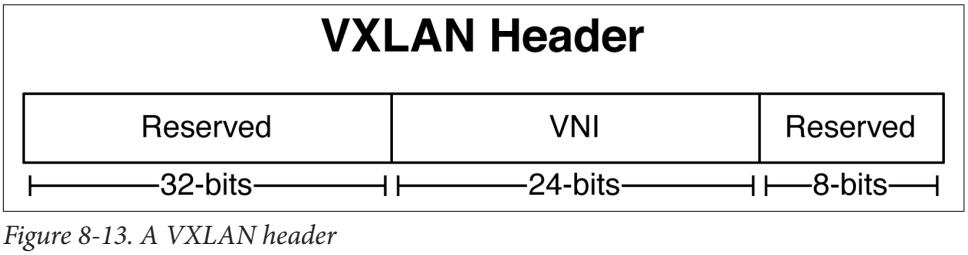


Figure 8-13. A VXLAN header

The first 32 bits of the VXLAN header are reserved. However, at least according to the IETF *draft-mahalingam-dutt-dcops-vxlan,* the fifth bit must have a value of 1 to indicate that it's a valid VXLAN header. Otherwise the other 31 bits have a value of 0.

The next 24 bits are the VNI. The VNI identifies the scope of the inner MAC frame that was originated by a VM. For example, you may create multiple VNIs with overlapping MAC addresses. To recap, you can think of a VNI as a bridge domain or VLAN. The good news is that we have twice the bits to play with when compared to traditional VXLANs. VLANs can support up to 2^{12} bridge domains. VXLAN tunnels can support up to 2^{24} or over 16 million bridge domains. Of course, this is the theoretical limit; real-world scaling numbers are dependent on the hardware and software used.

However, simply calculating the additional overhead of 64 bits for the VXLAN header doesn't tell the entire story when creating overlay tunnels across the network. Recall that you also need to transport this new VXLAN header across a Layer 3 network. A new outer Ethernet header, outer IP header, and outer UDP datagram is required to transport the VXLAN header across a Layer 3 network, as depicted in Figure 8-14.

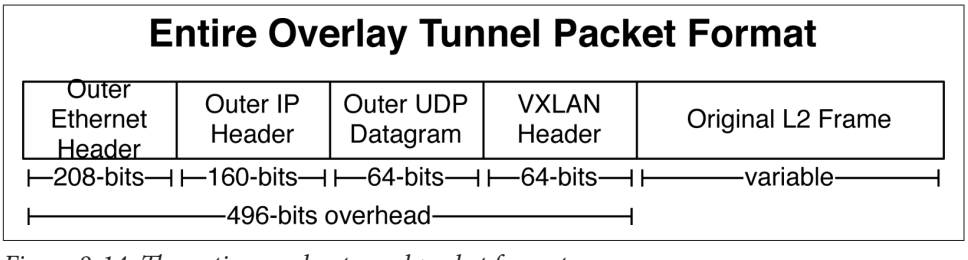


Figure 8-14. The entire overlay tunnel packet format

To sum up, an additional 496 bits of overhead is required to support a VXLAN tunnel between two VTEPs over a Layer 3 network. The total amount will vary slightly depending on the size of the original Layer 2 frame from the VM or physical server that's being encapsulated.

Because overlay networking requires additional overhead in a Layer 3 network, it's a great idea to enable jumbo frames throughout the entire network. Juniper QFX5100 switches support a Maximum Transmission Unit (MTU) of up to 9,216 bytes.

Overlay Controller

When using a unicast control plane protocol such as EVPN or OVSDB, a centralized overlay controller is required. At a high level, the overlay controller performs ingress replication for MAC learning when using a unicast control plane protocol. For example, with EVPN the overlay controller would speak MP-BGP to each VTEP as well as

provide BGP route reflection services to all of the other VTEPs to propagate new MAC addresses throughout the data center.

A controller is also required to provision VTEP across hypervisors and network switches. For example, every time a new hypervisor is added to the data center, it would require a management connection to the overlay controller. As new virtual networks are created, the overlay controller can create the appropriate VTEPs in the hypervisor and propagate MAC addresses.

The exception is using EVPN in a network-based VXLAN fabric, which does not require a centralized controller. Each switch will be running MP-BGP with family EVPN for MAC address learning. It's just like building an MPLS network, but the data plane encapsulation is VXLAN. The other note is that the VXLAN VNI is globally significant as opposed to MPLS labels which are locally significant.

Virtual Routers

Up to this point in the chapter, I have only covered switching and forwarding of Ethernet frames within the same bridge domain. If a VM needs to route to another VM in a different VNI, there needs to be a Layer 3 gateway address that the VM can use as a default router. Each VNI has a virtual router (vRouter) associated with it that VMs can use as a default gateway. The vRouter is depicted as "L3 GW" in Figure 8-15.

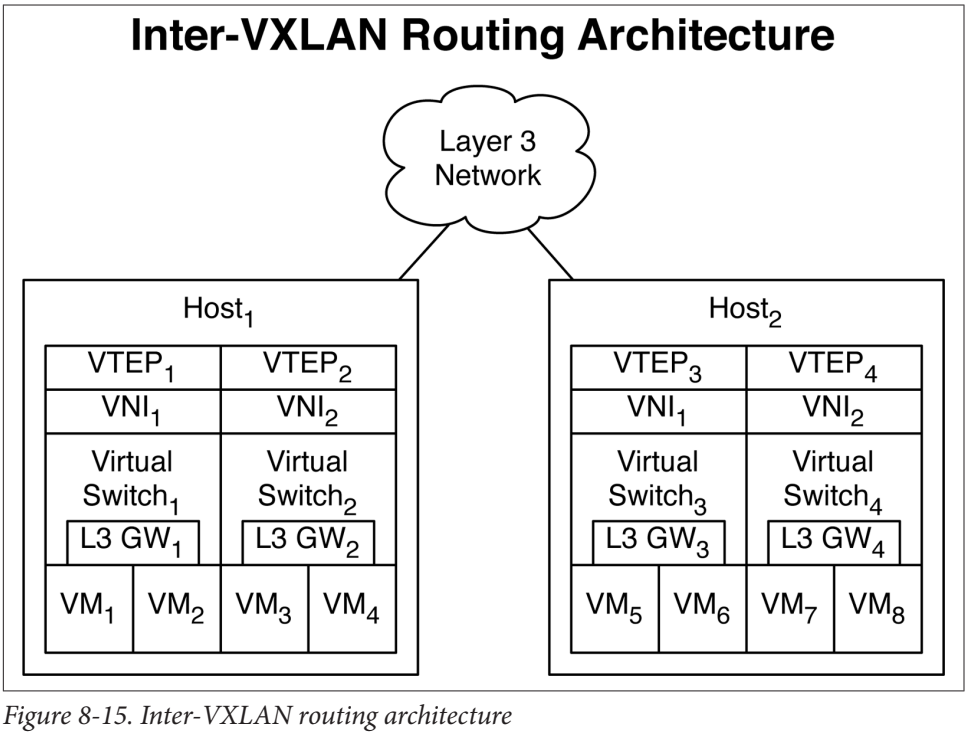


Figure 8-15. Inter-VXLAN routing architecture

Routing VM traffic

Let's study a simple example of VM_1 routing to VM_3. In this scenario, the VMs are in different VNI segments and require routing. We'll also make the assumption that VM_1 and VM_3 are in different networks, so that the operating system in VM_1 sees that the network address of VM_3 isn't part of the same subnet and must use the default gateway. VM_1 sends traffic to L3 GW_1; because the L3 GW_1 has full visibility of the VTEP lookup table, it sees that the destination MAC address of VM_3 is associated with $VTEP_2$. Because VM_1 and VM_3 are within the same host, there's no need to encapsulate the packet. The L3 GW_1 can simply locally route the traffic directly to VM_3.

The next example is if VM_1 needs to route traffic to VM_8. The same initial steps take place. VM_1 uses its default gateway of L3 GW_1. The L3 GW_1 receives the traffic and because it has full visibility into the VTEP tables, it sees that the destination IP address is owned by $VTEP_4$. The L3 GW_1 sends the traffic to $VTEP_1$ which encapsulates the packet and sends it off to $VTEP_4$. When the packet arrives at $VTEP_4$, it strips the packet of the VXLAN encapsulation and sees that the destination address is associated with a MAC address in Virtual $Switch_4$. The frame is forwarded through Virtual $Switch_4$ with a source MAC address of the L3 GW_4 and a destination MAC address of VM_8.

Routing physical traffic

Routing VM traffic between VNIs is fairly straightforward; it all happens in software with vRouters. The more difficult use case is routing physical servers in an overlay architecture. At a high level, there are two options:

Software-Based VXLAN Routing
: The physical server uses a default gateway that's located within a VM. This means that traffic from the physical server destined to the virtual default gateway must pass through overlay tunnels until it reaches the Service VM that owns the default gateway for that VNI. When the traffic reaches the Service VM, the same process applies as if it were a VM. The major drawback is that it creates asymmetric traffic flows for physical servers.

Hardware-Based VXLAN Routing
: Routing between VXLAN VNIs can happen in the networking hardware if the underlying equipment supports it. For example, the Juniper EX9200 and Juniper MX series routers support VXLAN routing. You simply create bridge domains and associate them with VNIs. Each bridge domain has a routed interface that is associated with an Integrated Routing and Bridging (IRB) interface that's able to route traffic between bridge domains. Because the VXLAN traffic is routed in hardware, there is no performance loss and servers can transmit at line rate.

Routing VXLAN traffic in the networking hardware also eliminates the asymmetric traffic patterns that exist in the software-based VXLAN routing solution, because the default gateways exist a single hop away in hardware. Also keep your eyes out for some new high-density Juniper QFX switches based on Juniper silicon that will support hardware.

The most preferable option would be to have a completely virtualized environment and not have to worry about routing physical traffic across an overlay architecture. The second best option would be to simply use the underlying networking equipment to route between VXLAN segments so that the virtual and physical servers operate in a seamless overlay architecture.

Storage

Many people assume that the storage must also take part in the overlay architecture because all of the VMs and physical servers require VTEPs to communicate with one another. However this is a common misconception. Making the assumption that the storage is IP-based, such as iSCSI or NFS, hypervisors and physical servers can simply use the underlying IP Fabric to reach the storage device directly over IP, as is demonstrated in Figure 8-16.

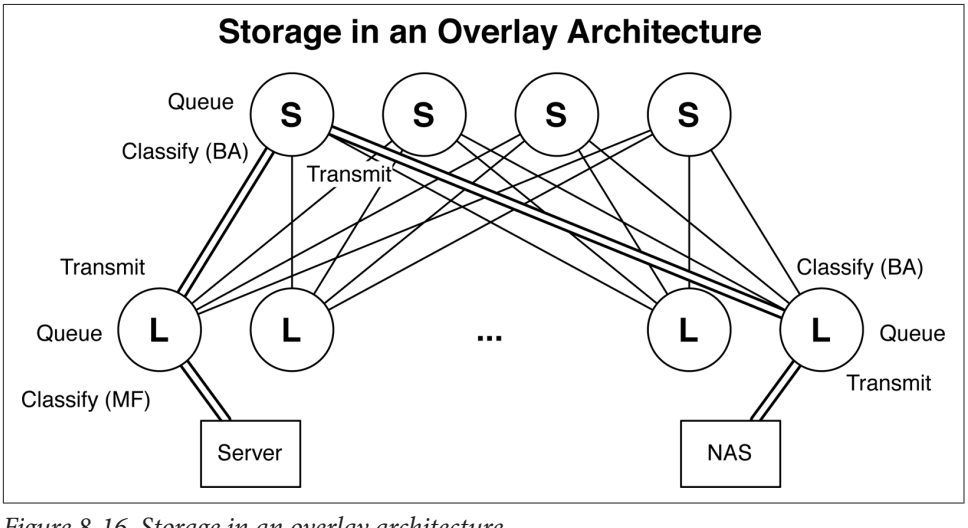


Figure 8-16. Storage in an overlay architecture

One of the benefits of using IP-based storage in an IP Fabric is that it's reachable from any location by using Layer 3. The server in Figure 8-16 needs to access the network-attached storage (NAS) and simply routes the traffic. Although the underlying network is an IP Fabric, you can still configure lossless Ethernet queues so that storage traffic is prioritized through the network, as shown in Figure 8-16. The path between

the server and NAS device is shown in a double-line in Figure 8-16. Here is what happens at each step along the way:

- The server transmits a packet destined to the NAS device.
- The first leaf switch receives the traffic and classifies the storage traffic by using a multifield classifier with a firewall filter.
- The leaf switch gives priority to the storage traffic and sends the traffic to the spine switch by using a high-priority IEEE 802.1p bit.
- The spine switch receives the traffic and identifies the high-priority storage traffic based on the IEEE 802.1p bit by using a behavior aggregate (BA).
- The spine switch gives priority to the storage traffic and sends the traffic to the destination spine switch by using a high-priority IEEE 802.1p bit.
- The destination leaf switch identifies the storage traffic with a BA, gives it priority, and transmits it to its final destination.

If your server and storage device support Priority-Based Flow Control (PFC)/IEEE 802.1Qbb, the Juniper QFX5100 series supports PFC over Layer 3, as well. The caveat is that PFC must operate over the IEEE 802.1p bits, therefore the point-to-point links between the spine and leaves must support IEEE 802.1Q. The result is that with lossless Ethernet queuing and PFC, servers and storage will always have guaranteed bandwidth and flow control in an overlay architecture based on an IP Fabric.

Juniper Architectures for Overlay Networks

Now that you have a better understanding of what problems an overlay architecture solves and how overlay networks operate, the next question is how can the Juniper QFX5100 family add additional value to an overlay network? Juniper QFX5100 switches support two key architectures to enable overlay architectures: Virtual Chassis Fabric (VCF) and an IP Fabric. Each architecture has its advantages and disadvantages. Most enterprise customers lean toward VCF because of its plug-and-play nature and because it's easy to manage. Large enterprises and service providers prefer to use an IP Fabric because it offers much larger scale and a seamless routing protocol with their existing environments.

We discussed VCF in detail in Chapter 5. The power of VCF increases when you combine it with an overlay networking architecture. The first thing to recall about VCF is that the routing engine is centralized and all of the other switches in the fabric are line cards. When you create a VTEP in a VCF, you do so in the configuration only once. However, the VTEP is programmed into every single switch in the VCF, saving you from having to configure numerous times throughout your network, as shown in Figure 8-17.

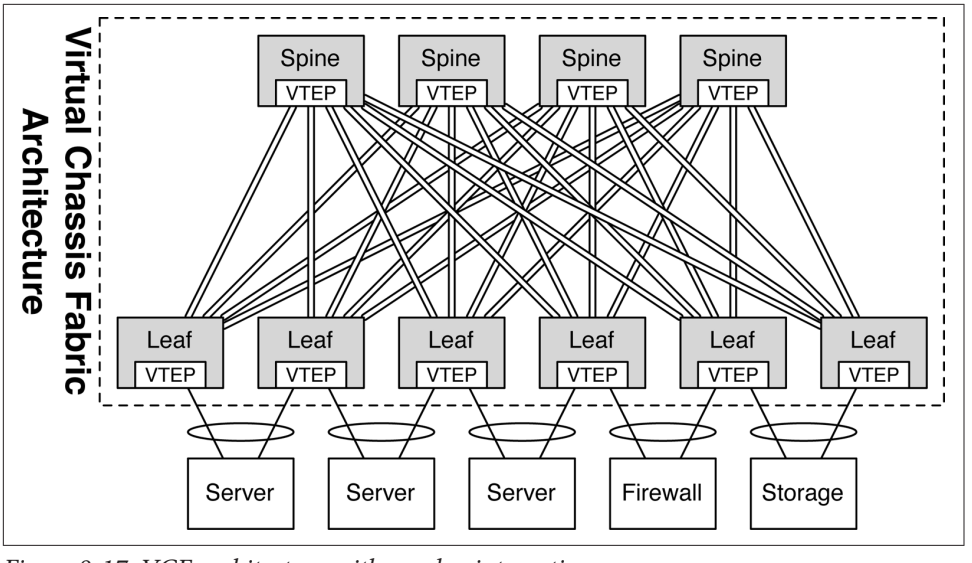


Figure 8-17. VCF architecture with overlay integration

It's also common that hypervisor hosts have multiple connections into the network. A common standard is to use 4 10GbE or 8 10GbE ports from the host into the network depending on the VM density of the host. Because VCF natively supports IEEE 802.3ad/LACP across multiple access switches, the host hypervisor can use standard LACP across all its uplinks into the network.

The really cool thing is that if there's a physical network that needs to be integrated into the overlay architecture and requires multihoming via IEEE 802.3ad/LACP, VCF can support it and also handle the VTEP and MAC learning across LACP on behalf of the server.

Of course, VCF supports ISSU. When you need to perform network maintenance and upgrade the software, you can do so without interrupting the traffic flow of the servers.

In brief, VCF is a great solution for enterprise customers looking to move toward an overlay architecture. It offers a single point of management, so the entire data center looks like a single logical switch. Physical servers can seamlessly be integrated into the overlay architecture and support multihoming.

Configuration

Let's jump right into the configuration details of overlay networking. There are many moving parts, as discussed in the architecture section. Let's take each component, one by one, and see how it works on the command line.

The assumption made in this configuration exercise is that we're integrating the Juniper QFX5100 switch into an existing VMware NSX environment running the OVSDB protocol.

Supported Hardware

First things first: let's review which Juniper hardware supports overlay networking with built-in VTEPs into the hardware. As of this writing, there are three platforms that support hardware accelerated VTEPs:

- Juniper QFX5100 series
- Juniper EX9200 series
- Juniper MX series

The Juniper QFX5100 family uses the Broadcom Trident II chipset and accommodates basic Layer 2–only VTEPs. Keep in mind that these VTEPs can only map the MAC addresses to an interface and aren't capable of VXLAN routing. The Juniper EX9200 and MX Series are based on purpose-built Juniper silicon and support mapping of MAC addresses, interfaces, and IP addresses, allowing you to both switch and route VXLAN traffic.

Controller

Let's make the assumption that you're configuring a controller-based overlay network architecture by using the OVSDB control plane protocol. The first step is to configure the controller's IP address:

```
[edit]
root# set protocols ovsdb controller 192.168.61.112
```

After you commit the configuration, the next step is to check the connection to the controller and see if it established an active connection:

```
dhanks@QFX5100> show ovsdb controller
VTEP controller information:
Controller IP address: 192.168.61.112
Controller protocol: ssl
Controller port: 6632
Controller connection: up
Controller seconds-since-connect: 56376
Controller seconds-since-disconnect: 0
Controller connection status: active
```

You can see that the connection to the VMware NSX controller is up and active.

Interfaces

The next step is to identify which interfaces are to be controlled by the OVSDB protocol. These are typically interfaces handling physical servers so that they can participate in the overlay architecture with the other VMs:

```
[edit]
root# set protocols ovsdb interfaces xe-0/0/3.0
```

Now, verify that the interface is managed by OVSDB:

```
dhanks@QFX5100> show ovsdb statistics interface
Interface Name: xe-0/0/3.0
Num of rx pkts:  0                    Num of tx pkts:  0
Num of rx bytes: 0                    Num of tx bytes: 0
```

Although there's no traffic flowing through the interface yet, you can see that the interface now appears in the `show ovsdb statistics` command.

Switch Options

The last step is to configure the switch at a global level to be managed by OVSDB. You also need to configure the source address to be used by the switch when talking to the OVSDB controller:

```
[edit]
root# set switch-options ovsdb-managed
[edit]
root# set switch-options vtep-source-interface lo0.0
```

With the basics configured, let's move on to the verification.

Logical Switch

Because you set a global knob called `ovsdb-managed`, the VMware NSX controller is able to dynamically create logical switches on the Juniper QFX5100 switch. Take a look at the logical switches on the switch:

```
dhanks@QFX5100> show ovsdb logical-switch
Logical switch information:
Logical Switch Name: 4918d9c6-0ac2-444f-b819-d763d577b099
Flags: Created by both
VNI: 100
Num of Remote MAC: 5
Num of Local MAC: 0
```

Notice the dynamically created logical switch name that ensures uniqueness across the network. This particular logical switch is associated with VNI and has already learned five remote MAC addresses.

Remote MACs

To view what MAC addresses were learned by the controller, use the following command:

```
dhanks@QFX5100> show ovsdb mac remote
Logical Switch Name: 4918d9c6-0ac2-444f-b819-d763d577b099
  Mac                 IP              Encapsulation      Vtep
  Address             Address                            Address
  40:b4:f0:07:97:f0   0.0.0.0         Vxlan over Ipv4    100.100.120.120
  64:87:88:ac:42:0d   0.0.0.0         Vxlan over Ipv4    100.100.120.120
  64:87:88:ac:42:18   0.0.0.0         Vxlan over Ipv4    100.100.130.130
  a8:d0:e5:5b:5f:08   0.0.0.0         Vxlan over Ipv4    100.100.130.130
  ff:ff:ff:ff:ff:ff   0.0.0.0         Vxlan over Ipv4    100.100.100.1
```

You can see that five MAC addresses have been learned. Recall that the Juniper QFX5100 series is based on the Broadcom Trident II chipset and is unable to see the IP address, because the VTEP table only holds the MAC address and interface.

OVSDB Interfaces

You can also verify that the OVSDB-managed interfaces are associated to the correct bridge domains by using the following command:

```
root@QFX5100> show ovsdb interface
Interface             VLAN ID         Bridge-domain
xe-0/0/3.0            0               4918d9c6-0ac2-444f-b819-d763d577b099
```

You can see that the `xe-0/0/3.0` interface is correctly associated with the dynamically created bridge domain from the VMware NSX controller.

VTEPs

One of the more interesting commands that you can use is to discover what other VTEPs exist in the network. Recall that we learned five MAC addresses from the VMware NSX controller. Let's take a look and see how many MAC addresses are associated with remote VTEPs:

```
dhanks@QFX5100> show ovsdb virtual-tunnel-end-point
Encapsulation              Ip Address                      Num of MAC's
VXLAN over IPv4            100.100.100.1                   1
VXLAN over IPv4            100.100.120.120                 2
VXLAN over IPv4            100.100.130.130                 2
```

You can see that a single MAC address came from the VTEP 100.100.100.1, whereas you received two MAC addresses from the VTEPs 100.100.120.120 and 100.100.130.130.

Switching Table

The last step is to see the Ethernet switching table. We can verify the MAC addresses and see which logical interface to which they're being forwarded:

```
dhanks@QFX5100> show ethernet-switching table

MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC)


Ethernet switching table : 4 entries, 0 learned
Routing instance : default-switch
  Vlan                MAC                 MAC         Age    Logical
  name                address             flags              interface
  4918d9c6-0ac2-444f-b819-d763d577b099 40:b4:f0:07:97:f0 SO       - vtep.32769
  4918d9c6-0ac2-444f-b819-d763d577b099 64:87:88:ac:42:0d SO       - vtep.32769
  4918d9c6-0ac2-444f-b819-d763d577b099 64:87:88:ac:42:18 SO       - vtep.32770
  4918d9c6-0ac2-444f-b819-d763d577b099 a8:d0:e5:5b:5f:08 SO       - vtep.32770
```

Each of the five MAC addresses are being forwarded within the same bridge domain, but to different VTEPs, as shown in the `Logical interface` column.

Multicast VTEP Exercise

Now that you have an idea how to handle a controller-based overlay configuration, let's move to a controller-less architecture using multicast. In this exercise, you'll set up the topology in Figure 8-18.

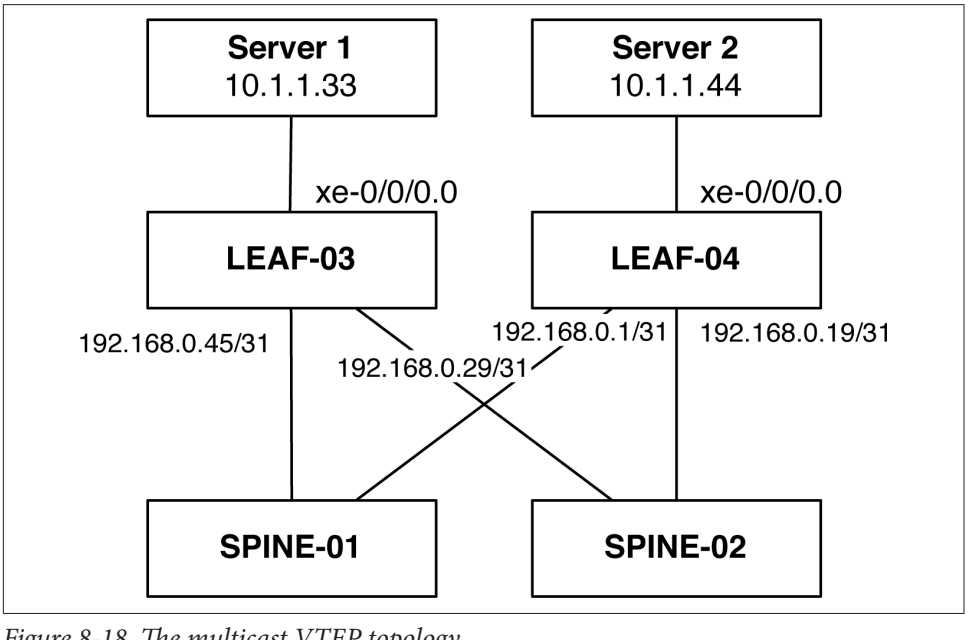


Figure 8-18. The multicast VTEP topology

The assumption is that there is a pure Layer 3 network with multicast enabled to which LEAF-03 and LEAF-04 are connected. The trick is to make Server 1 and Server 2 talk to each other over Layer 2 through a VXLAN tunnel that traverses the Layer 3

network. If Server 1 can ping Server 2 through a VXLAN tunnel, the test will be considered successful.

LEAF-03 Configuration

The first step is to configure the server-facing interface on LEAF-01 as an access port in the `foobar` VLAN and configure a loopback address for the switch:

```
interfaces {
    xe-0/0/0 {
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members foobar;
                }
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.7/32;
            }
        }
    }
```

Next, configure a static VTEP and use the loopback address as its source address:

```
switch-options {
    vtep-source-interface lo0.0;
}
```

Finally, define the `foobar` VLAN and set up the VXLAN VNI:

```
vlans {
    foobar {
        vlan-id 100;
        vxlan {
            vni 100;
            multicast-group 225.10.10.10;
        }
    }
}
```

Notice that we have the ability to perform VLAN normalization. As long as the VNI stays the same between remote VTEPs, the VLAN handoff can be any value back to the server. Because we're using multicast for the MAC learning, we need to associate the VNI with a multicast group.

LEAF-04

The configuration for LEAF-04 is identical to that of LEAF-03:

```
interfaces {
    xe-0/0/0 {
```

```
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members foobar;
                }
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.5/32;
            }
        }

}
switch-options {
    vtep-source-interface lo0.0;
}
vlans {
    foobar {
        vlan-id 100;
        vxlan {
            vni 100;
            multicast-group 225.10.10.10;
        }
    }
}
```

Verification

Take a moment to use some `show` commands to verify that you configured the basics correctly:

```
{master:0}
dhanks@temp-leaf-03> show vlans

Routing instance        VLAN name             Tag          Interfaces
default-switch          foobar                100
                                                           vtep.32769*
                                                           xe-0/0/0.0*
```

The VLAN is there and associated with the correct interface. You can also see that the VTEP is successfully created and associated with the `foobar` VLAN, as well.

Now, double-check your VTEP and ensure that it's associated with the correct VLAN and multicast group:

```
{master:0}
dhanks@temp-leaf-03> show ethernet-switching vxlan-tunnel-end-point source
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx
<default>                 0   10.0.0.7         lo0.0    0
    L2-RTT                   Bridge Domain              VNID     MC-Group-IP
    default-switch           foobar+100                 100      225.10.10.10
```

Very cool! You see that your `foobar` bridge domain is bound to the VNI 100 with the correct multicast group. You can also see that your source VTEP is configured with the local loopback address of 10.0.0.7.

Take a look at what remote VTEPs have been identified:

```
{master:0}
root@temp-leaf-03> show ethernet-switching vxlan-tunnel-end-point remote
Logical System Name       Id  SVTEP-IP         IFL   L3-Idx
<default>                 0   10.0.0.7         lo0.0    0
 RVTEP-IP         IFL-Idx   NH-Id
 10.0.0.5         567       1757
    VNID          MC-Group-IP
    100           225.10.10.10
```

This is great. You can see the remote VTEP on LEAF-04 as 10.0.0.5.

Now, check the Ethernet switching table and see if the two servers have exchanged MAC addresses yet:

```
{master:0}
dhanks@temp-leaf-03> show ethernet-switching table

MAC flags (S - static MAC, D - dynamic MAC, L - locally learned, P - Persistent
static
           SE - statistics enabled, NM - non configured MAC, R - remote PE MAC)


Ethernet switching table : 2 entries, 2 learned
Routing instance : default-switch
    Vlan                MAC                 MAC         Age    Logical
    name                address             flags              interface
    foobar              f4:b5:2f:40:66:f8   D             -    xe-0/0/0.0
    foobar              f4:b5:2f:40:66:f9   D             -    vtep.32769
```

Excellent! There is a local (f4:b5:2f:40:66:f8) and remote (f4:b5:2f:40:66:f9) MAC address in the switching table!

The last step is to ping from Server 1 to Server 2. Given that the VTEPs have discovered each other and the servers have already exchanged MAC addresses, you can be pretty confident that the ping should work:

```
bash# ping -c 10.1.1.44
PING 10.1.1.44 (10.1.1.44): 56 data bytes
64 bytes from 10.1.1.44: icmp_seq=0 ttl=64 time=1.127 ms
64 bytes from 10.1.1.44: icmp_seq=1 ttl=64 time=1.062 ms
64 bytes from 10.1.1.44: icmp_seq=2 ttl=64 time=1.035 ms
64 bytes from 10.1.1.44: icmp_seq=3 ttl=64 time=1.040 ms
64 bytes from 10.1.1.44: icmp_seq=4 ttl=64 time=1.064 ms

--- 10.1.1.44 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.035/1.066/1.127/0.033 ms
```

Indeed. Just as expected. With the tunnel successfully created, bound to the correct VLAN, and the MAC addresses showing up in the switching table, the ping has

worked successfully. Server 1 is able to ping Server 2 across the same Layer 2 network through a VXLAN tunnel that's traversing a Layer 3 network. The VTEPs on the switches are using multicast for MAC address learning, which simulates a physical Layer 2 switch characteristics to flood, filter, and forward broadcast and unknown unicast Ethernet traffic.

Summary

This chapter thoroughly discussed overlay networking in the data center. There are two categories of overlay architectures: controller-based and controller-less. The controller-based architectures use control plane–based protocols such as OVSDB (VMware NSX) and EVPN (Juniper Contrail) for MAC address learning. The controller-less options include multicast for data plane learning and MP-BGP with EVPN.

We reviewed the Juniper products that support overlay architectures and how they vary. The Juniper QFX5100 series, which is based on the Broadcom Trident II chipset, only supports Layer 2 VTEPs. The Juniper EX9200 and MX Series, which are based on the Juniper Trio chipset, offer both Layer 2 and Layer 3 VTEPs with which you can switch and route overlay traffic.

We walked through two configuration examples of overlay architectures. The first was an assumption of an IP Fabric with VMware NSX that uses the OVSDB protocol. We configured the controller, VTEP, interfaces, and walked through the commands to verify the setup. The final exercise was using a controller-less architecture that used multicast for MAC learning between the statically defined VTEPs. We were able to ping between two servers across a Layer 2 VXLAN tunnel across a Layer 3 IP Fabric.

CHAPTER 9
Network Analytics

One of the most difficult tasks in network operations is gathering accurate sampling data from a switch to create a dashboard that shows the overall health of the network. Accurate network visibility and analytics is the cornerstone to operating an efficient and reliable network. After all, how do you know that your network is running smoothly if you have no idea what's going across it?

Network analytics is a broad term, but in general—as network operators—we want to provide context and answer the following questions:

- What types of applications are consuming network resources?
- What's the current capacity and utilization of a given switch?
- How can I quickly identify peaks and valleys?
- How can I detect microbursts?
- Are there hotspots forming in the network?

Answering these questions has become more difficult with the standardization of 10GbE access ports in the data center. The amount of traffic is increasing rapidly and traditional sampling techniques such as sFlow and IPFIX only provide answers to some of the questions posed. Because microbursts and latency spikes can happen in very small windows, tools that rely on sampling every few seconds are unable to detect these events that interrupt business applications. Microburst events occur when there are multiple ports of ingress traffic that's all destined to a single egress port, and the egress port's buffer is exceeded. For example, if server 1 sent a query to a set of compute clusters, and all 100 compute clusters responded back to the server at the exact same time, the physical port connected to server 1 would become congested for that brief moment in time.

To detect micro events in the network, the frequency at which the networking device samples the traffic and counters must be increased dramatically. With Juniper Enhanced Analytics, you can receive real-time information from the switch and detect events such as latency, jitter, and microbursting.

Overview

The Juniper QFX5100 series gives you the capability to quickly gather traffic statistics and other data out of the switch and into powerful collection tools so that you can visualize what's happening inside the network (see Figure 9-1). Juniper QFX5100 switches supports two major types of network analytics:

Sampled Data
: The sFlow technology on the Juniper QFX5100 family uses sampling to gather data. You can sample interface statistics and flow data on a Juniper QFX5100 switch at a frequency of one out of *n* packets. Data is exported from the Juniper QFX5100 every 1,500 bytes or every 250 ms. Due to the nature of sampling, there are no options to enable monitoring thresholds; this means you're unable to send real-time alerts based on events exceeding or dropping below a threshold.

Real-Time Data
: Juniper Enhanced Analytics fills in the gaps of traditional sampling techniques such as sFlow. Data is exported from the switch in real time as the data is collected. Enhanced Analytics offers much faster polling intervals, all the way down to 8 ms. Because data is collected in real time, you are able to set high and low thresholds for latency and queue depth, all the way down to 1 nanosecond.

One of the benefits of sFlow is that it's able to capture the first 128 bytes of the sampled packets. It's a small form of Deep Packet Inspection (DPI), which remote tools can use to create detailed graphs of the application traffic within the network. Although Enhanced Analytics doesn't have any (current) DPI capabilities, it has the unique ability to detect micro events and report them in real time. By combining the power of sampled and real-time data, you can get a true end-to-end view of what's happening within your network.

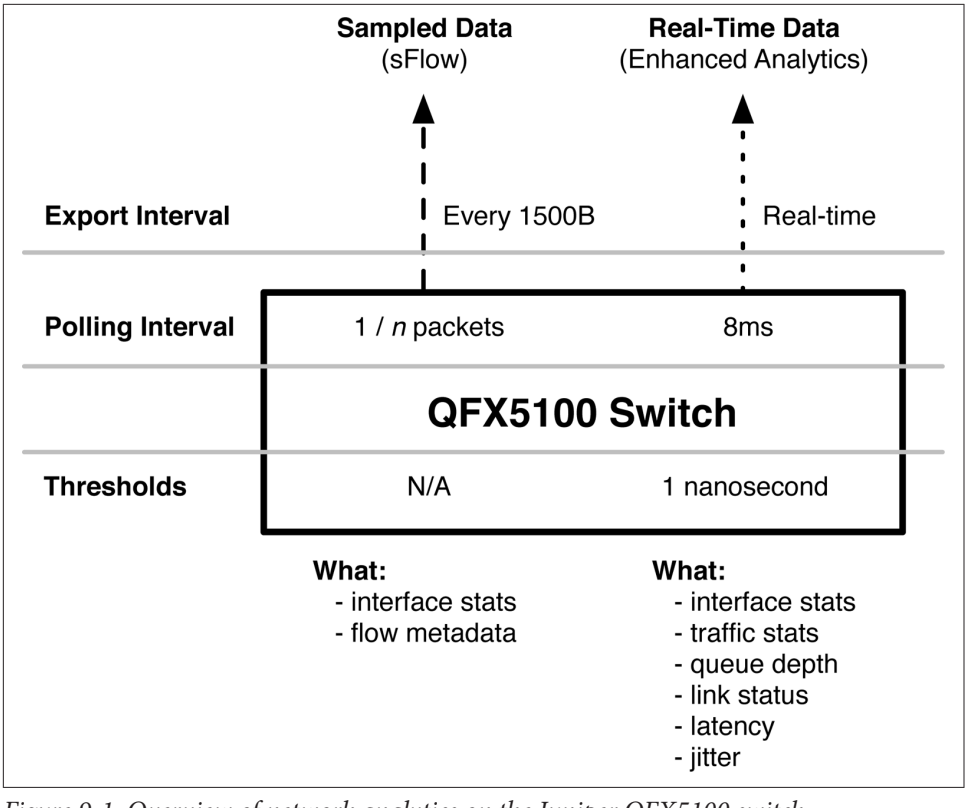


Figure 9-1. Overview of network analytics on the Juniper QFX5100 switch

sFlow

Figure 9-2 shows at a high level how sFlow collects samples of packets in a switched network and sends the aggregated data to a remote collector.

There are two sampling mechanisms for sFlow:

Packet-Based Sampling
: You can sample one packet out of a specified number of packets from a particular interface. The first 128 bytes—including the raw Ethernet frame—are recorded and sent to the collector. It's important to note that only switched traffic can be subject to sFlow; you cannot sample Layer 3 interfaces. The data included in the sampled information is the aforementioned Ethernet frame, IP packet, TCP segments or UDP datagrams, and any remaining payload information up to 128 bytes.

Time-Based Sampling
: Using this mode, you can capture interface statistics at a specified time interval to the remote collector. If you don't need to sample packets and get the first 128 bytes of information but instead only want to receive traffic statistics, time-based sampling would be a good option for you.

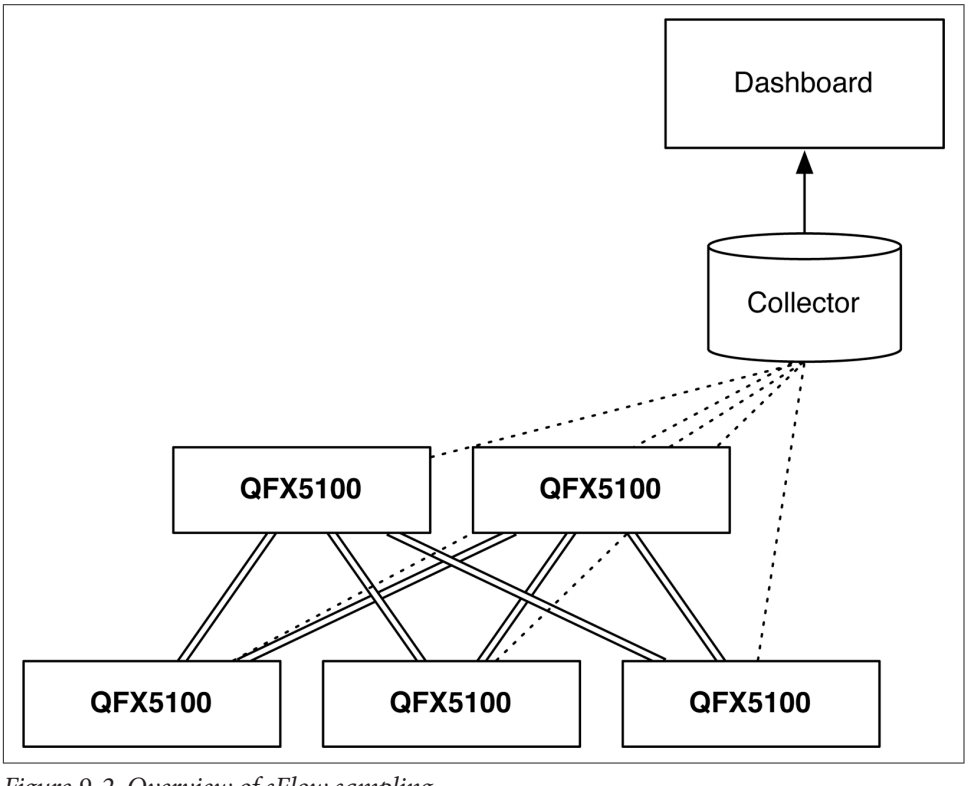


Figure 9-2. Overview of sFlow sampling

sFlow is commonly used to enable network dashboards using collection tools such as PRTG or nfsen. It shows what types of applications are consuming networking resources. Because the first 128 bytes of the packet are sent to the collector, it can easily perform DPI into the payload of the packet and see what's happening from an application perspective.

Adaptive Sampling

As you might imagine, enabling sFlow across all interfaces in a switch that could support 104 10GbE interfaces would require a lot of processing to sample packets, perform DPI, and send that data to an external collector. You wouldn't want sFlow to cause any service interruptions to the actual traffic itself. Juniper sFlow includes the

capability to monitor the interface traffic and dynamically adjust the polling interval of sFlow.

Agents check the interfaces every 5 seconds and create a sorted list of interfaces that are receiving the most samples per second. The top five interfaces with the highest number of samples are selected. Using a binary backoff algorithm, the sampling loads on the selected interfaces are reduced by half and allocated to other interfaces that have a lower sampling rate. Keep in mind that adaptive sampling is a transient feature that's adjusted every 5 seconds. If traffic spiked for 1 minute and then went back down for the next 15 minutes, the adaptive sampling would kick in for the first minute, but then restore sFlow to the configured values for the remaining 15 minutes. Sampling resources are distributed evenly across the entire switch during excessive traffic peaks, resulting in the guaranteed delivery of production traffic through the switch.

Configuration

Be aware that an external collection tool is required to make sFlow useful. Downloading and installing an external collection tool is beyond the scope of this book and is left as an exercise to the user. However some of the better sFlow tools are Juniper's STRM, PRTG, ntop, nfsen, and sFlowTrend.

The first step in the configuration process is to set up the sFlow collector to which we want to send the sampled data. The Juniper QFX5100 series supports sending data from the management port as well as any revenue ports configured for Layer 3. It's recommended to use revenue ports to export sampled data, because during peak traffic loads, the amount of data being exported can be quite large.

Let's get right to it:

```
{master:0}[edit]
dhanks@QFX5100# set protocols sflow collector 192.168.1.100 udp-port 5000
```

Next, define which interfaces will be enabled for sFlow sampling. By default all the interfaces are excluded from sFlow, and you must enable them for sFlow to work:

```
{master:0}[edit]
dhanks@QFX5100# set protocols sflow interfaces et-0/0/0
```

The final step is to set up the polling interval and sampling rate for the interfaces. You can define these settings per interface or simply set them globally:

```
{master:0}[edit]
dhanks@QFX5100# set protocols sflow sample-rate egress 10 ingress 10
{master:0}[edit]
dhanks@QFX5100# set protocols sflow polling-interval 5
```

You might be wondering what the difference is between the `polling-interval` and the `sampling-rate`; these two knobs are often confused. The `polling-interval` simply instructs the Juniper QFX5100 device to poll the physical interface every *n* sec-

onds to collect interface statistics. The `sampling-rate` specifies how many packets the Juniper QFX5100 switch inspects in order to send sampled meta-information (the first 128 bytes) to the collector.

sFlow Review

The Juniper QFX5100 family of switches supports sFlow for all switched traffic passing through the switch. It allows you to quickly get an idea of what types of applications are consuming networking resources. There are only a few configuration statements to enable sFlow and it's very easy to get running. However, there are a few caveats, which are listed here:

- You cannot enable sFlow on Layer 3 interfaces or aggregated Ethernet bundles (`ae`); however, you can enable sFlow on the member interfaces such as `et-0/0/0`.
- When using sFlow on ingress traffic, none of the CPU-bound traffic is captured.
- When using sFlow on egress traffic, no multicast or broadcast traffic is sampled. Also the Juniper QFX5100 device doesn't factor in egress firewall filters when using sFlow, due to a limitation in the Broadcom chipset.
- The Juniper QFX5100 series supports sFlow version 5 as of 13.1X51D20.

Using sFlow is a great way to quickly sample application traffic in your network and visualize it. After enabling sFlow, many network operators are surprised to learn what types of applications and end-user traffic is going across the network.

Enhanced Analytics

With the introduction of 10GbE and higher speeds in the access layer, new use cases have emerged such as Big Data and High-Frequency Trading (HFT). Each of these requires high-speed networks, low latency, and no jitter. Traditional monitoring tools such as sFlow aren't equipped to deal with the high-speed latency and jitter problems that can arise in high-speed networks. This is because sFlow works by sampling traffic. For example if sFlow only sampled one packet out of 2,000, it wouldn't be able to detect a microburst happening in the other 1,999 packets.

Overview

With Juniper Enhanced Analytics, you can monitor the Juniper QFX5100 in real time (as opposed to sampling packets) to monitor traffic statistics, queue depth, latency, and jitter in the network (see Figure 9-3). Being able to collect real-time traffic statistics offers more granularity when graphing traffic patterns across interfaces. The queue depth and latency are early warning signals to application failures. For example, if you notice an increasing amount of tail-dropping or microbursts on a specific

server, you know that it will have a negative impact on the application performance and reliability.

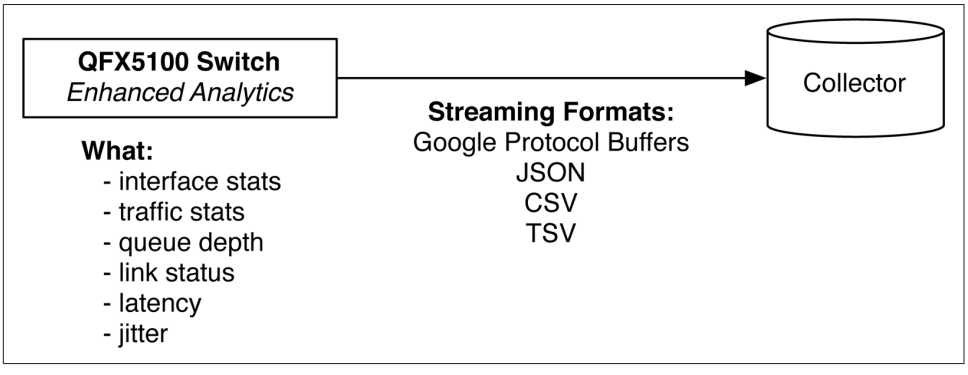


Figure 9-3. Overview of Juniper Enhanced Analytics

Enhanced Analytics is split into two major functions; the result is that a Juniper QFX5100 device is able to quickly export data to multiple collectors in real time for offline analysis. Following is a brief description of each function:

Analytics Daemon
: The analytics daemon (analyticsd) runs within Junos; its primary responsibility is to collect the analytics information from the Packet Forwarding Engine (PFE) and export it to the collectors.

Analytics Manager
: The analytics manager (AM) runs within the PFE so that it's able to read traffic, queue depth, and latency in real time. Traffic is read off the data plane and processed into ring buffers so that analyticsd can retrieve the information.

Enhanced Analytics and sFlow make a perfect combination when you need to quickly get all the data off the switch and into offline analysis tools. You get both the benefits of sampled and real-time data to create a true end-to-end view of your network.

Architecture

Both analyticsd (AD) and AM work as a team to obtain real-time data from the PFE and export it to remote collectors, as shown in Figure 9-4.

The two analytics engines, AD and AM, work in unison and use standard Unix Interprocess Communication (IPC) to pass information back and forth. The heavy lifting is performed by the Junos μKernel. Traffic statistics are gathered from the Broadcom chipset every second, and the queue depth information is retrieved every 8 ms (see Figure 9-5). The information is put into ring buffers that the IPC thread uses to retrieve the information; the traffic statistics are pulled from the ring queue every sec-

ond, and the queue depth is pulled from the ring queue every 100 ms. The rest of the processing is handled by the control plane with the analytics daemon. AD uses standard IPC to transfer data from the AM. From this point the data is shipped off to the configured collectors.

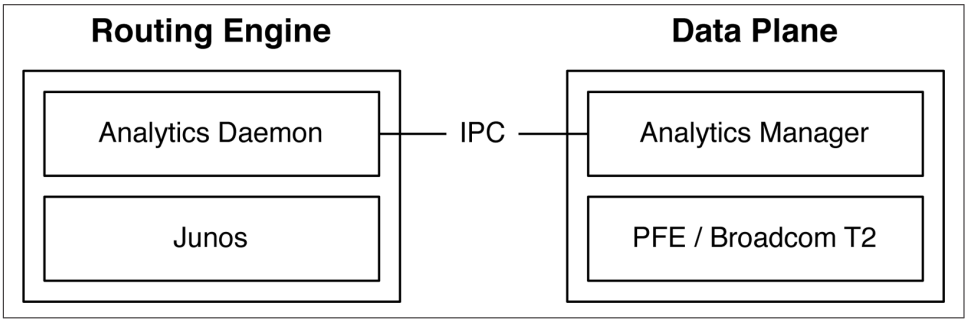


Figure 9-4. Analytics daemon and analytics manager overview

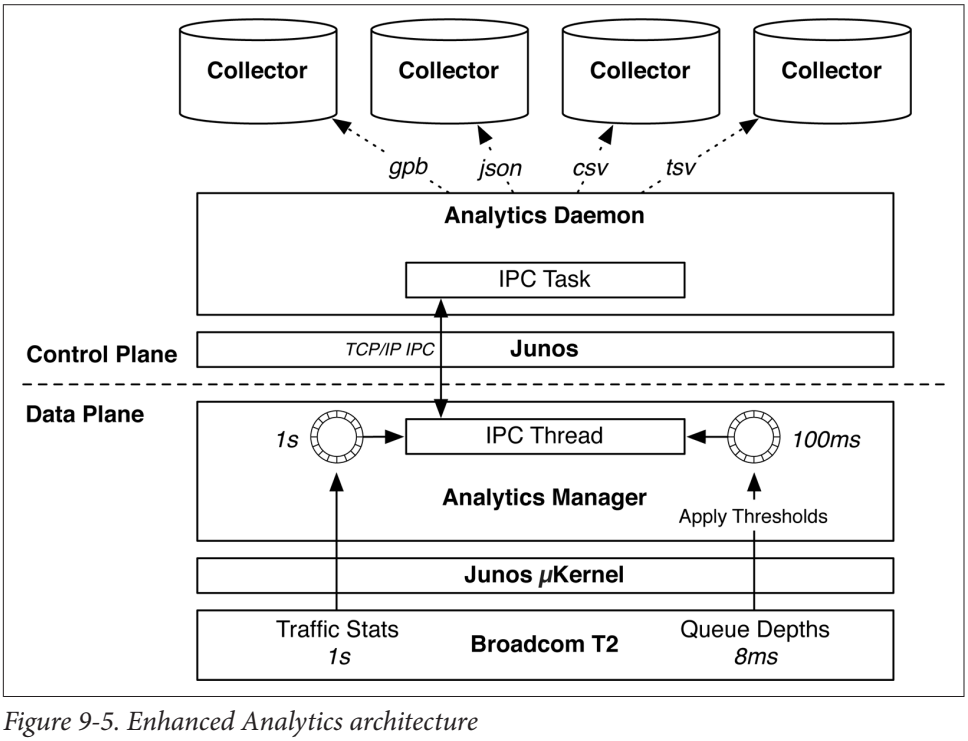


Figure 9-5. Enhanced Analytics architecture

The end result is that data is retrieved from the data plane in real time and exported to multiple collectors. The information gathered makes it possible for you to quickly determine the overall network performance, health, and application stability.

The Enhanced Analytics architecture shown in Figure 9-5 is accurate as of Junos 13.1X51D20. Given that the entire architecture is a software solution, it can be changed and enhanced at any time with future software releases.

Streaming Information

The information provided by Enhanced Analytics is critical in mapping out your network to detect latency and jitter between applications. The information streamed is divided into two major categories:

Streamed Queue Depth Statistics
: You can use the queue depth information to measure an interface's latency and see how full the transmit buffer is. If the buffer capacity is exceeded, traffic will drop.

Streamed Traffic Statistics
: You can use the traffic statistics to see the amount and velocity of traffic flowing through the network. The information also includes any types of errors and dropped packets.

Using the combination of queue depth and traffic statistics, you can quickly troubleshoot application issues in your data center. The extensive support for streaming protocols reduces the burden to create customized monitoring tools and increases the compatibility with open source tools, such as LogStash, fluentd, and Elasticsearch.

Streaming formats

Enhanced Analytics is capable of streaming the queue depth and traffic information to multiple collectors in the following streaming formats:

Google Protocol Buffer
: The Google Protocol Buffer (GPB) supports nine types of messages in a hierarchical format. The format is in binary and isn't readable by humans, unless you're Cypher from *The Matrix*.

JavaScript Object Notation (JSON)
: JSON is a lightweight data-interchange format that is easy for both humans and machines to read and parse. It's based on a subset of the JavaScript Programming Language.

Comma-Separated Values (CSV)
: This is a simple flat file containing fields of data delimited by a single comma (",").

Tab-Separated Values (TSV)
: Simple flat file containing fields of data delimited by a single tab ("\t").

Each format has its advantages and disadvantages. If you need quick and dirty, you might opt for the CSV or TSV formats. If you really enjoy programming in Python or Perl, you might like to use the JSON format. If you need sheer speed and support for remote procedure calls (RPCs), you might lean toward GPB.

GPB. Take a moment to examine the GPB format, as presented in Table 9-1.

Table 9-1. GPB streaming format specifications

Byte position	Field
0 to 3	Length of message
4	Message version
5 to 7	Reserved

The Juniper QFX5100 family uses a specific GPB prototype file (*analytics-proto*) to format the streaming data, which you can download from the Juniper website.

Let's take a look at the fields of the *analytics-proto* file. This is what you will need to use in your GPB collector:

```
package analytics;

// Traffic statistics related info
message TrafficStatus {
    optional uint32             status          = 1;
    optional uint32             poll_interval   = 2;
}

// Queue statistics related info
message QueueStatus {
    optional uint32             status          = 1;
    optional uint32             poll_interval   = 2;
    optional uint64             lt_high         = 3;
    optional uint64             lt_low          = 4;
    optional uint64             dt_high         = 5;
    optional uint64             dt_low          = 6;
}

message LinkStatus {
    optional uint64             speed           = 1;
    optional uint32             duplex          = 2;
    optional uint32             mtu             = 3;
    optional bool               state           = 4;
    optional bool               auto_negotiation= 5;
}

message InterfaceInfo {
    optional uint32             snmp_index      = 1;
    optional uint32             index           = 2;
    optional uint32             slot            = 3;
    optional uint32             port            = 4;
    optional uint32             media_type      = 5;
    optional uint32             capability      = 6;
```

```
    optional uint32             porttype        = 7;
}

message InterfaceStatus {
    optional LinkStatus         link            = 1;
    optional QueueStatus        queue           = 2;
    optional TrafficStatus      traffic         = 3;
}

message QueueStats {
    optional uint64             timestamp       = 1;
    optional uint64             queue_depth     = 2;
    optional uint64             latency         = 3;
    optional string             traffic_class   = 4;
}

message TrafficStats {
    optional uint64             timestamp       = 1;
    optional uint64             rxpkt           = 2;
    optional uint64             rxucpkt         = 3;
    optional uint64             rxmcpkt         = 4;
    optional uint64             rxbcpkt         = 5;
    optional uint64             rxpps           = 6;
    optional uint64             rxbyte          = 7;
    optional uint64             rxbps           = 8;
    optional uint64             rxdrop          = 9;
    optional uint64             rxerr           = 10;
    optional uint64             txpkt           = 11;
    optional uint64             txucpkt         = 12;
    optional uint64             txmcpkt         = 13;
    optional uint64             txbcpkt         = 14;
    optional uint64             txpps           = 15;
    optional uint64             txbyte          = 16;
    optional uint64             txbps           = 17;
    optional uint64             txdrop          = 18;
    optional uint64             txerr           = 19;
}

//Interface message
message Interface {
    required string             name            = 1;
    optional bool               deleted         = 2;
    optional InterfaceInfo      information     = 3;
    optional InterfaceStatus    status          = 4;
    optional QueueStats         queue_stats     = 5;
    optional TrafficStats       traffic_stats   = 6;
}

message SystemInfo {
    optional uint64             boot_time       = 1;
    optional string             model_info      = 2;
    optional string             serial_no       = 3;
    optional uint32             max_ports       = 4;
    optional string             collector       = 5;
    repeated string             interface_list  = 6;
}

message SystemStatus {
    optional QueueStatus        queue           = 1;
```

```
    optional TrafficStatus      traffic         = 2;
}

//System message
message System {
    required string             name            = 1;
    optional bool               deleted         = 2;
    optional SystemInfo         information     = 3;
    optional SystemStatus       status          = 4;
}
```

JSON. Following are two examples of JSON. The first example will be queue depth information:

```
{"record-type":"queue-stats","time":1383453988263,"router-id":"qfx5100-switch",
"port":"xe-0/0/18","latency":0,"queue-depth":208}
```

The next example is traffic statistics:

```
{"record-type":"traffic-stats","time":1383453986763,"router-id":"qfx5100-switch",
"port":"xe-0/0/16","rxpkt":26524223621,"rxpps":8399588,"rxbyte":3395100629632,
"rxbps":423997832,"rxdrop":0,"rxerr":0,"txpkt":795746503,"txpps":0,"txbyte":1018555
33467, "txbps":0,"txdrop":0,"txerr":0}
```

CSV. Now, let's explore CSV, using the same data as last time. First up is the queue depth information:

```
q,1383454067604,qfx5100-switch,xe-0/0/18,0,208
```

Here are the traffic statistics:

```
t,1383454072924,qfx5100-switch,xe-
0/0/19,1274299748,82950,163110341556,85603312,0,0,
  27254178291,8300088,3488534810679,600002408,27268587050,3490379142400
```

TSV. Finally we have TSV. It's the exact same thing as CSV, but uses a tab (\t) instead of a comma (",") for a delimiter. First up is the queue depth information:

```
Q   585870192561703872    qfx5100-switch    xe-0/0/18       (null)  208     2
```

You get the idea. There's no need to show you the traffic statistics.

Streamed queue depth information

The streamed queue depth information is straightforward and makes it possible for you to easily see each interface's buffer utilization and latency. Table 9-2 lists the data collected in detail.

Table 9-2. Streamed queue depth output fields

Field	Description
`record-type`	The type of statistic. Displayed as the following: • `queue-stats` (JSON) • `q` (CSV or TSV)
`time`	The time at which the information was captured. The format is Unix epoch, which is the number of seconds/microseconds since January 1, 1970.
`router-id`	IPv4 `router-id` of the source switch.
`port`	Name of the physical port.
`latency`	Traffic queue latency in milliseconds.
`queue-depth`	Depth of the queue in bytes.

Streamed traffic information

The streamed traffic information has a very similar format to the queue depth information. Take a look at each of the fields, as shown in Table 9-3.

Table 9-3. Streamed traffic statistics output fields

Field	Description
`record-type`	The type of statistic. Displayed as the following: • `traffic-stats` (JSON) • `t` (CSV or TSV)
`time`	The time at which the information was captured. The format is Unix epoch.
`router-id`	IPv4 `router-id` of the source switch.
`port`	Name of the physical port.
`rxpkt`	Total packets received.
`rxpps`	Total packets received per second.
`rxbyte`	Total bytes received.
`rxbps`	Total bytes received per second.
`rxdrop`	Total incoming packets dropped.
`rxerr`	Total incoming packets with errors.
`txpkt`	Total packets transmitted.
`txpps`	Total packets transmitted per second.
`txbyte`	Total bytes transmitted.
`txbps`	Total bytes transmitted per second.
`txdrop`	Total transmitted packets dropped.
`txerr`	Total transmitted packets with errors.

Configuration

The configuration of Enhanced Analytics is very modular in nature. At a high level, there are resources that reference `resource-profiles` and there are collectors that reference `export-profiles`, as shown in Figure 9-6.

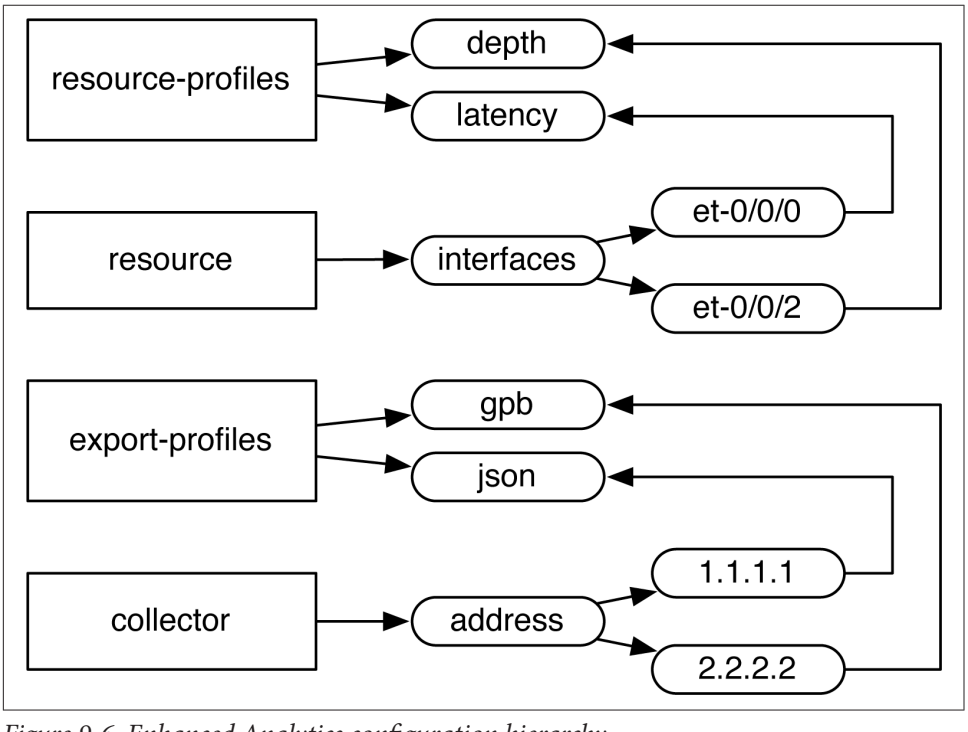


Figure 9-6. Enhanced Analytics configuration hierarchy

Because the configuration is modular in nature, you can create a single Enhanced Analytics configuration that contains multiple profiles for different applications, collectors, and streaming formats. Changing the way a Juniper QFX5100 switch performs analytics is as simple as changing a profile, which triggers all of the underlying changes such as collector addressing, streaming formats, latency thresholds, and traffic statistics.

Let's inspect the full configuration that's illustrated in Figure 9-6. We'll define the following:

- Two resource profiles
- Two export profiles
- Monitor two interfaces

- Create two collectors with different streaming formats

Here is the code:

```
services {
    analytics {
        traceoptions {
            file an size 10m files 3;
        }
        export-profiles {
            GPB {
                stream-format gpb;
                interface {
                    information;
                    statistics {
                        traffic;
                        queue;
                    }
                    status {
                        link;
                        traffic;
                        queue;
                    }
                }
                system {
                    information;
                    status {
                        traffic;
                        queue;
                    }
                }
            }
            JSON {
                stream-format json;
                interface {
                    information;
                    statistics {
                        traffic;
                        queue;
                    }
                    status {
                        link;
                        traffic;
                        queue;
                    }
                }
                system {
                    information;
                    status {
                        traffic;
                        queue;
                    }
                }
            }
        }
        resource-profiles {
            QUEUE-DEPTH-STANDARD {
                queue-monitoring;
                traffic-monitoring;
```

```
                depth-threshold high 14680064 low 1024;
            }
            LATENCY {
                queue-monitoring;
                traffic-monitoring;
                latency-threshold high 900000 low 100;
            }
        }
        resource {
            system {
                resource-profile QUEUE-DEPTH-STANDARD;
                polling-interval {
                    traffic-monitoring 2;
                    queue-monitoring 100;
                }
            }
            interfaces {
                et-0/0/0 {
                    resource-profile QUEUE-DEPTH-STANDARD;
                }
                et-0/0/1 {
                    resource-profile LATENCY;
                }
            }
        }
        collector {
            local {
                file an.local size 10m files 3;
            }
            address 1.1.1.1 {
                port 3000 {
                    transport udp {
                        export-profile GPB;
                    }
                }
            }
            address 2.2.2.2 {
                port 5555 {
                    transport tcp {
                        export-profile JSON;
                    }
                }
            }
        }
    }
}
```

It's just like building blocks. You define a set of attributes and then reference it in another part of the configuration. This makes Enhanced Analytics a breeze.

There are some tricks that you need to be aware of when configuring the queue depth thresholds. The values are given in bytes, but it's all relative to the physical interface being monitored.

Here are the calculations for latency, given in bytes for different speed interfaces:

- 1GbE latency = bytes / 125
- 10GbE latency = bytes / 1250
- 40GbE latency = bytes / 5000

So, for example if you were monitoring a 10GbE interface and wanted to detect 1µ of latency, you would set the number of bytes to 1250.

The next step is using `show` commands to verify that Enhanced Analytics is set up correctly. Verify the collectors first:

```
{master:0}
dhanks@QFX5100> show analytics collector
Address        Port   Transport  Stream format  State           Sent
1.1.1.1        3000   udp        gpb            n/a             8742
2.2.2.2        5555   tcp        json           Established     401
```

Everything looks great. Obviously, the User Datagram Protocol (UDP) collector says "N/A" because UDP is a stateless protocol and the switch doesn't have any acknowledgements whether the traffic was received.

Now, let's take a look at the general analytics configuration:

```
{master:0}
dhanks@QFX5100> show analytics configuration
Traffic monitoring status is enabled
Traffic monitoring polling interval : 2 seconds
Queue monitoring status is enabled
Queue monitoring polling interval : 100 milliseconds
Queue depth high threshold : 14680064 bytes
Queue depth low threshold : 1024 bytes

Interface     Traffic    Queue      Queue depth            Latency
            Statistics Statistics    threshold            threshold
                                   High       Low        High      Low
                                   bytes)               (nanoseconds)
et-0/0/0    enabled    enabled     14680064   1024       n/a       n/a
et-0/0/1    enabled    enabled     n/a        n/a        900000    100
```

Looking good. Both interfaces are configured for traffic and queue depth information with the correct thresholds. The traffic monitoring polling is set correctly at every two seconds. The queue monitoring polling interval is correct per Figure 9-5.

Take a peek at the information the Juniper QFX5100 device is gathering around traffic statistics:

```
{master:0}
dhanks@QFX5100> show analytics traffic-statistics
CLI issued at 2014-07-26 20:40:43.067972
Time: 00:00:01.905564 ago, Physical interface: et-0/0/1
  Traffic Statistics:                Receive           Transmit
  Total octets:                  633916936          633662441
  Total packets:                   8703258            8699671
  Unicast packet:                  8607265            8603658
  Multicast packets:                 94802              94810
  Broadcast packets:                  1191               1203
  Octets per second:                  2048               1704
  Packets per second:                    3                  3
  CRC/Align errors:                      0                  0
  Packets dropped:                       0                  0
Time: 00:00:01.905564 ago, Physical interface: et-0/0/0
  Traffic Statistics:                Receive           Transmit
  Total octets:                  633917501          633662336
  Total packets:                   8703209            8699607
  Unicast packet:                  8607214            8603571
  Multicast packets:                 94819              94831
  Broadcast packets:                  1176               1205
  Octets per second:                  1184               1184
  Packets per second:                    2                  2
  CRC/Align errors:                      0                  0
  Packets dropped:                       0                  0
```

Very cool! There's no need to log in to a collector to confirm that the Juniper QFX5100 is configured correctly to gather traffic statistics. We can view it locally with the `show analytics traffic-statistics` command. The really great thing is that the command-line output has microsecond precision.

Summary

This chapter covered network analytics and how you can use the built-in tools to create a better performing and more reliable network. Network analytics comes in two forms: sampled data and real-time data. The sampled data is performed by sFlow; the real-time data is performed by Enhanced Analytics. The sFlow technology allows you to quickly take a peek inside your switching network and see application-level information. It's always surprising to see what type of traffic is flowing through a network. With Enhanced Analytics, you can get precision data in terms of traffic statistics, latency, and queue depth information in real time. Finally you learned that the Juniper QFX5100 series of switches supports multiple streaming formats: GPB, JSON, CSV, and TSV.

APPENDIX A

Under the Hood

One of the most frustrating things about networking technology is that oftentimes operators are caged into a box called the command-line interface (CLI). Anything behind the curtain doesn't exist and isn't supported. Unfortunately, I can't change the not-supported part, but I can at least show you what's behind the curtains and how it works.

Any additional information you're able to pull from a piece of technology ultimately makes your network better, whether it's better network management, graphing, or troubleshooting.

Big Scary Disclaimer

Everything I'm about to show you in this chapter isn't supported by Juniper Networks or the Juniper Technical Assistance Center (JTAC). Don't use these commands in production. Use them at your own risk. Changing any values at a low level will cause instability in the network because the changes will not be synchronized with the control plane.

With that out of the way, let's get on with having some fun!

The Broadcom Shell

The Broadcom shell, owned and maintained by Broadcom, is the standard CLI that you can use to directly access the Broadcom chipsets. It is a simple tool that you can put to work gathering additional debugging information from the system.

Messing around with the Broadcom shell is really powerful but at the same time really dangerous to production systems. Many of the commands are simply not documented for the average user. If you want the full documentation of the commands and command output, it's required that you be part of the Broadcom NDA. Generally, that's reserved for vendors such as Juniper Networks and very, very, very large customers who have a business need to get low-level access. The official documentation for the Broadcom tables and registers is well over 9,000 pages in length.

Be warned. With great power comes great responsibility. Try to stick with simple `show` commands. I highly recommend that you never use commands to write values directly to the Broadcom chip for two simple reasons:

1. You have no idea what you're doing. You don't have the documentation. Moreover, even if you did, you wouldn't be reading this chapter as a reference anyway.
2. Any changes you make to the Broadcom chipset are not synchronized with Junos; they will be out of sync. Things break.

Overview

There are three types of primary data structures in the Broadcom chipset:

Tables
: Tables contain a set of views.

Views
: Views contain a structured data.

Registers
: Registers contains key-value pairs.

I will briefly walk you through each step on how to get data from each of the three types of data stores.

Tables

The first step to learning the Junos µkern is logging in:

```
dhanks@QFX5100:RE:0% vty fpc0

TOR platform (1500Mhz Pentium processor, 255MB memory, 0KB flash)

TFXPC0(vty)#
```

The next step is to determine how to list what tables exist within the Broadcom chipset. The good news is that's easy:

```
TFXPC0(vty)# set dcbcm bcmshell "listmem"

HW (unit 0)
 Flags   Name                  Entry/Copy Description
 ----bC  ALTERNATE_EMIRROR_BITMAP  256    Source Modid based blocking mask table
 ----bC  BCAST_BLOCK_MASK        107      Broadcast Block Mask, FeatureSpeci...
 -----C  COS_MAP_SEL             107      Select one of four sections of COS...
 --A-bC  CPU_COS_MAP             128      index by COPYTO_CPU reasons code a...
 -----C  CPU_COS_MAP_DATA_ONLY   128      CPU_COS_MAP Data SRAM for CPU_COS_...
 --A-bC  CPU_COS_MAP_ONLY        128      CPU_COS_MAP TCAM only view
 -----C  CPU_PBM                   1      Specifies the port(s) that is (are...
 -----C  CPU_PBM_2                 1      Specifies the port(s) that is (are...
 ----bC  CPU_TS_MAP              256      vlan range match table
 ----bC  DEST_TRUNK_BITMAP      1024      Destination Trunk Bitmap Table.
```

The bad news is that there are nearly 900 tables. Don't forget that each table has multiple views.

Views

Now that you know there are nearly 900 tables to play around with, let's see how many views one of them has. A really good table to look at to see the IPv4 Forwarding Information Base (FIB) is the `L3_ENTRY` table:

```
TFXPC0(vty)# set dcbcm bcmshell "listmem L3_ENTRY"

HW (unit 0)
 Flags   Name                  Entry/Copy Description
 ------  L3_ENTRY_HIT_ONLY     36864      L3 Hit bit table
 ------  L3_ENTRY_HIT_ONLY_X   36864      L3 Hit bit table, FeatureSpecific-...
 ------  L3_ENTRY_HIT_ONLY_Y   36864      L3 Hit bit table, FeatureSpecific-...
 --h--C  L3_ENTRY_IPV4_MULTICAST73728      L3 routing table IPV4 MULTICAST view
 --h--C  L3_ENTRY_IPV4_UNICAST 147456      L3 routing table IPV4 UNICAST view
 --h--C  L3_ENTRY_IPV6_MULTICAST36864      L3 routing table IPV6 MULTICAST view
 --h--C  L3_ENTRY_IPV6_UNICAST 73728      L3 routing table IPV6 UNICAST view
 ------  L3_ENTRY_LP           36864      L3_ENTRY LP Control Table.
 --h-b-  L3_ENTRY_ONLY         147456      L3 routing table with fb_regs arch...
Flags: (r)eadonly, (d)ebug, (s)orted, (h)ashed
       C(A)M, (c)bp, (b)ist-able, (C)achable
```

The `L3_ENTRY` table has nine views. Take a glance at the data inside the view `L3_ENTRY_IPV4_UNICAST`:

```
TFXPC0(vty)# set dcbcm bcmshell "dump chg L3_ENTRY_IPV4_UNICAST"

HW (unit 0)
L3_ENTRY_IPV4_UNICAST.ipipe0[82656]:
<VRF_ID=1,VALID=1,NEXT_HOP_INDEX=0x10e,KEY=0x0021400000e0,IP_ADDR=0xa000007,IPV4UC:
VRF_ID=1,IPV4UC:NEXT_HOP_INDEX=0x10e,IPV4UC:KEY=0x0021400000e0,IPV4UC:IP_ADDR=0xa00
0007,IPV4UC:HASH_LSB=7,IPV4UC:ECMP_PTR=0x10e,IPV4UC:DATA=0x21c000,HASH_LSB=7,ECMP_P
TR=0x10e,DATA=0x21c000,>
L3_ENTRY_IPV4_UNICAST.ipipe0[84164]:
<VRF_ID=1,VALID=1,NEXT_HOP_INDEX=0x10c,KEY=0x003815000360,IP_ADDR=0xc0a8001b,IPV4UC
:VRF_ID=1,IPV4UC:NEXT_HOP_INDEX=0x10c,IPV4UC:KEY=0x003815000360,IPV4UC:IP_ADDR=0xc0
```

```
a8001b,IPV4UC:HASH_LSB=0x1b,IPV4UC:ECMP_PTR=0x10c,IPV4UC:DATA=0x218000,HASH_LSB=0x1
b,ECMP_PTR=0x10c,DATA=0x218000,>
```

A lot of the data is in hexadecimal and requires conversion to be human readable. For an example, look at the following key-value pair:

```
IP_ADDR=0xa000007
```

The IP address would need to be translated from hexadecimal into dotted decimal. In this example, the human-readable IP address would be 10.0.0.7.

Registers

The final place to poke around and find data is in the registers. There are over 3,800 registers; it would certainly require a lot of time to look at each of them and make heads and tails of any of it.

The first step is to simply find out what registers exist:

```
TFXPC0(vty)# set dcbcm bcmshell "listreg -s *"

HW (unit 0)
 g3---  ARB_RAM_DBGCTRL            ipipe0    ARB_RAM_CONTROL
 p3---  ASF_PORT_CFG               mmu0      ASF_PORT_SPEED
 g3---  AUX_ARB_CONTROL            ipipe0    IP auxilary arbiter control re...
 g3---  AUX_ARB_CONTROL_2          ipipe0    IP auxilary arbiter control re...
 ?3---  AXI_SRAM_MEMC_CONFIG       cmic0     AXI SRAM MEMC Configuration - ...
 g3---  BFD_RX_ACH_TYPE_CONTROL0   ipipe0    Stores the ACH Channel Type va...
 g3---  BFD_RX_ACH_TYPE_CONTROL1   ipipe0    Stores the ACH Channel Type va...
 g3---  BFD_RX_ACH_TYPE_MPLSTP     ipipe0    Stores the ACH Channel Types f...
 g6---  BFD_RX_ACH_TYPE_MPLSTP1    ipipe0    Stores the ACH Channel Types f...
 g3---  BFD_RX_UDP_CONTROL         ipipe0    UDP destination Port number fo...
 g3---  BFD_RX_UDP_CONTROL_1       ipipe0    UDP destination Port number fo...
 g6--r  BKPMETERINGDISCSTATUS0     mmu0      PORT_BITMAP: Current Back Pres...
 g6--r  BKPMETERINGDISCSTATUS1     mmu0      PORT_BITMAP: Current Back Pres...
 g6--r  BKPMETERINGWARNSTATUS0     mmu0      PORT_BITMAP: Current Back Pres...
 g6--r  BKPMETERINGWARNSTATUS1     mmu0      PORT_BITMAP: Current Back Pres...
 g3---  BST_HW_SNAPSHOT_EN         mmu0      Enable Buffer Statistics Track...
 g3---  BST_SNAPSHOT_ACTION_EN     mmu0      Enable reset BST_TRACKING_ENAB...
 g3---  BST_TRACKING_ENABLE        mmu0      Enable Buffer Statistics Track...
```

There are no views associated with registers; you can pull the data directly from them. A good register to look at is `CPU_CONTROL_1`:

```
TFXPC0(vty)# set dcbcm bcmshell "getreg CPU_CONTROL_1"

HW (unit 0)
CPU_CONTROL_1.ipipe0[1][0x3a000300]=0x8524020: <VXLT_MISS_TOCPU=0,
   V6L3ERR_TOCPU=0,V6L3DSTMISS_TOCPU=0,V4L3ERR_TOCPU=1,V4L3DSTMISS_TOCPU=0,
   UUCAST_TOCPU=0,URPF_MISS_TOCPU=0,UMC_TOCPU=0,TUNNEL_ERR_TOCPU=0,
   STATICMOVE_TOCPU=0,SRCROUTE_TOCPU=0,RESERVED_7=0,RESERVED_21=0,
   RESERVED_19=0,RESERVED_16=0,NONSTATICMOVE_TOCPU=0,NIP_L3ERR_TOCPU=0,
   MC_INDEX_ERROR_TOCPU=0,MARTIAN_ADDR_TOCPU=1,L3_SLOWPATH_TOCPU=1,
   L3_MTU_FAIL_TOCPU=1,L3UC_TTL_ERR_TOCPU=1,L3UC_TTL1_ERR_TOCPU=1,
   IPMC_TTL_ERR_TOCPU=0,IPMC_TTL1_ERR_TOCPU=0,IPMCPORTMISS_TOCPU=0,
   IPMCERR_TOCPU=0,HG_HDR_TYPE1_TOCPU=0,HG_HDR_ERROR_TOCPU=0,FCOE_DST_MISS_TOCPU=0,
   CLASS_BASED_SM_PREVENTED_TOCPU=0>
```

The CPU_CONTROL_1 register shows you how many host packets were sent to the CPU because of a failure.

Broadcom Shell and cprod

You can also use the cprod command to execute the Broadcom shell commands. If you want to dump a list of the tables and views to the FreeBSD file system, use the following:

```
root@temp-spine-02:RE:0% time cprod -A fpc0 -c 'set dcbcm bcmshell "listmem"' >
/tmp/listmem
0.000u 0.004s 0:00.50 0.0%      0+0k 0+0io 0pf+0w
```

How did I know that there are nearly 900 tables? Just use wc:

```
root@temp-spine-02:RE:0% wc -l /tmp/listmem
     892 /tmp/listmem
```

Same thing, but for the registers:

```
root@temp-spine-02:RE:0% time cprod -A fpc0 -c 'set dcbcm bcmshell "listreg -s *"'
> /tmp/listreg
0.006u 0.006s 0:02.00 0.0%      0+0k 0+2io 0pf+0w
root@temp-spine-02:RE:0% wc -l /tmp/listreg
    3845 /tmp/listreg
```

If you wanted to get a little bit fancy, you can begin parsing the data we saved from the tables:

```
root@temp-spine-02:RE:0% cat /tmp/listmem | awk '{print $2}' | tail -5
VLAN_XLATE_LP
VOQ_COS_MAP
VOQ_MOD_MAP
VOQ_PORT_MAP
VRF
XLPORT_WC_UCMEM_DATA
```

The astute reader could then write a shell or Python script to cycle through all of the tables, find all of the views, and then dump all of the *table.view* data.

Summary

This chapter introduced the Broadcom shell. Although it was littered with warnings, the overall intent of the chapter was to give you additional tools for retrieving data from the QFX5100 series of switches. Although I made every attempt to show relevant Broadcom Shell commands throughout each chapter, there are cases for which you might need to dig a bit deeper. If you're a truly fearless network operator, I showed you how to get lost in the Broadcom shell. If you ever hit a scenario for which Junos doesn't provide you with enough low-level information and you have nothing but time on your hands, you could always find it buried somewhere within the Broadcom shell.

APPENDIX B

Optical Guide

Juniper Data Center Optics Matrix

Table B-1 presents a listing of Juniper optics and their attributes.

Table B-1. Juniper data center optics matrix

Type	Model	Description
10GbE optical SFP	QFX-SFP-10GE-ZR	10GBASE-ZR, SMF, 80 km
	QFX-SFP-10GE-ER	10GBASE-ER, SMF, 40 km
	QFX-SFP-10GE-LR	10GBASE-LR, SMF, 10 km
	QFX-SFP-10GE-SR	10GBASE-SR, MMF, 300 m
	QFX-SFP-10GE-USR	10GBASE-USR, MMF, 100 m
10GbE copper SFP	QFX-SFP-DAC-1M	Direct attach copper 10GbE to 10GbE, 1 m passive
	QFX-SFP-DAC-3M	Direct attach copper 10GbE to 10GbE, 3 m passive
	QFX-SFP-DAC-5M	Direct attach copper 10GbE to 10GbE, 5 m passive
	EX-SFP-10GE-DAC-7M	Direct attach copper 10GbE to 10GbE, 7 m passive
	QFX-SFP-DAC-1MA	Direct attach copper 10GbE to 10GbE, 1 m active
	QFX-SFP-DAC-3MA	Direct attach copper 10GbE to 10GbE, 3 m active
	QFX-SFP-DAC-5MA	Direct attach copper 10GbE to 10GbE, 5 m active
	QFX-SFP-DAC-7MA	Direct attach copper 10GbE to 10GbE, 7 m active
	QFX-SFP-DAC-10MA	Direct attach copper 10GbE to 10GbE, 10 m active
1GbE optical SFP	QFX-SFP-1GE-LX	1GE-LX, SMF, 10 KM
	QFX-SFP-1GE-SX	1GE-SX, MMF, 500 m
	QFX-SFP-1GE-T	1GE-T, Cat5e, 100 m

Type	Model	Description
40GbE optical QSFP	JNP-QSFP-40G-LR4	40G-LR4, SMF, 10 km
	QFX-QSFP-40G-ESR4	40G-ESR4, MMF, 400 m
	QFX-QSFP-40G-SR4	40G-SR4, MMF, 300 m
40GbE copper QSFP	QFX-QSFP-DAC-1M	Direct attach copper 40GbE to 40GbE, 1 m
	QFX-QSFP-DAC-3M	Direct attach copper 40GbE to 40GbE, 3 m
	JNP-QSFP-DAC-5M	Direct attach copper 40GbE to 40GbE, 5 m
	JNP-QSFP-DAC-5MA	Direct attach copper 40GbE to 40GbE, 5 m active
	JNP-QSFP-DAC-7MA	Direct attach copper 40GbE to 40GbE, 7 m active
	JNP-QSFP-DAC-10MA	Direct attach copper 40GbE to 40GbE, 10 m active
	QFX-QSFP-DACBO-1M	Direct attach copper 40GbE to 4 10GbE, 1 m
	QFX-QSFP-DACBO-3M	Direct attach copper 40GbE to 4 10GbE, 3 m
	JNP-QSFP-DACBO-5MA	Direct attach copper 40GbE to 4 10GbE, 5 m active
	JNP-QSFP-DACBO-7MA	Direct attach copper 40GbE to 4 10GbE, 7 m active
	JNP-QSFP-DACBO-10M	Direct attach copper 40GbE to 4 10GbE, 10 m active

Juniper Optics Compatibility Matrix

Table B-2 presents a listing of the Juniper optics and the Juniper switches with which they're compatible.

Table B-2. Juniper optics compatibility matrix

Type	Model	QFX3500 QFX3600	QFabric	QFX5100	EX4300	EX4500	EX4550	EX9200
10 GbE optical SFP	QFX-SFP-10GE-ZR			✓	✓	✓	✓	✓
	QFX-SFP-10GE-ER	✓	✓	✓	✓	✓	✓	✓
	QFX-SFP-10GE-LR	✓	✓	✓	✓	✓	✓	✓
	EX-SFP-10GE-LRM				✓	✓	✓	✓
	QFX-SFP-10GE-SR	✓	✓	✓	✓	✓	✓	✓
	JNP-10G-SR-8PACK	✓	✓	✓	✓	✓	✓	✓
	QFX-SFP-10GE-USR	✓	✓	✓	✓	✓	✓	✓

Type	Model	QFX3500 QFX3600	QFabric	QFX5100	EX4300	EX4500	EX4550	EX9200
10GbE copper SFP	QFX-SFP-DAC-1M	✓	✓	✓	✓	✓	✓	
	QFX-SFP-DAC-3M	✓	✓	✓	✓	✓	✓	
	QFX-SFP-DAC-5M	✓	✓	✓	✓	✓	✓	
	EX-SFP-10GE-DAC-7M				✓	✓	✓	
	QFX-SFP-DAC-1MA	✓	✓	✓				
	QFX-SFP-DAC-3MA	✓	✓	✓				
	QFX-SFP-DAC-5MA	✓	✓	✓				
	QFX-SFP-DAC-7MA	✓	✓	✓				
	QFX-SFP-DAC-10MA	✓	✓	✓				
1GbE optical SFP	QFX-SFP-1GE-LX	✓	✓	✓	✓	✓	✓	✓
	JNP-1G-SX-8PACK	✓	✓	✓	✓	✓	✓	✓
	QFX-SFP-1GE-SX	✓	✓	✓	✓	✓	✓	✓
	JNP-1G-T-8PACK	✓	✓	✓	✓	✓	✓	✓
	QFX-SFP-1GE-T	✓	✓	✓	✓	✓	✓	✓
40GbE optical QSFP	JNP-QSFP-40G-LR4	✓	✓	✓	✓		✓	✓
	QFX-QSFP-40G-ESR4	✓	✓	✓	✓			
	JNP-40G-SR4-4PACK	✓	✓	✓	✓		✓	✓
	QFX-QSFP-40G-SR4	✓	✓	✓	✓		✓	✓
40GbE copper QSFP	EX-QSFP-40GE-DAC-50cm				✓			
	QFX-QSFP-DAC-1M	✓	✓	✓	✓	✓	✓	
	QFX-QSFP-DAC-3M	✓	✓	✓	✓	✓	✓	
	JNP-QSFP-DAC-5M			✓	✓	✓	✓	
	JNP-QSFP-DAC-5MA	✓	✓	✓				
	JNP-QSFP-DAC-7MA	✓	✓	✓				
	JNP-QSFP-DAC-10MA	✓	✓	✓				
	QFX-QSFP-DACBO-1M	✓	✓	✓				
	QFX-QSFP-DACBO-3M	✓	✓	✓				
	JNP-QSFP-DACBO-5MA	✓	✓	✓				
	JNP-QSFP-DACBO-7MA	✓	✓	✓				
	JNP-QSFP-DACBO-10M	✓	✓	✓				

APPENDIX C

BGP and VTEP Configurations

In the exercises in Chapter 7 and Chapter 8, we configured an IP Fabric and a basic multicast overlay network, respectively. The full switch configurations are posted in this appendix for your reference.

For more information and to download these configurations directly, please visit our GitHub repository at *https://github.com/Juniper/qfx5100-book*.

LEAF-03

Here is the full Junos configuration of LEAF-03:

```
## Last commit: 2014-07-28 19:28:10 PDT by root
version "14.1-20140727_rt2_53_vjqfd.0 [dc-builder]";
/*
 * dhcpd-generated /var/etc/dhcpd.options.conf
 * Version: JDHCPD release 13.2X51-D20.2 built by builder on 2014-04-29 09:09:04
UTC
 * Written: Mon Jul 28 19:50:10 2014
 */
system {
    host-name temp-leaf-03;
    time-zone America/Los_Angeles;
    services {
        ssh {
            root-login allow;
            max-sessions-per-connection 32;
        }
        netconf {
            ssh;
        }
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
```

```
            any notice;
            authorization info;
        }
        file interactive-commands {
            interactive-commands any;
        }
    }
    extensions {
        providers {
            juniper {
                license-type juniper deployment-scope commercial;
            }
            chef {
                license-type juniper deployment-scope commercial;
            }
        }
    }
    processes {
        dhcp-service {
            traceoptions {
                file dhcp_logfile size 10m;
                level all;
                flag all;
            }
        }
        app-engine-virtual-machine-management-service {
            traceoptions {
                level notice;
                flag all;
            }
        }
    }
}
interfaces {
    interface-range ALL-SERVER {
        member xe-0/0/*;
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members SERVER;
                }
            }
        }
    }
    xe-0/0/0 {
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members foobar;
                }
            }
        }
    }
    et-0/0/48 {
        mtu 9216;
        unit 0 {
            description facing_spine-01;
```

```
        family inet {
            mtu 9000;
            address 192.168.0.45/31;
        }
    }
}
et-0/0/49 {
    mtu 9216;
    unit 0 {
        description facing_spine-02;
        family inet {
            mtu 9000;
            address 192.168.0.29/31;
        }
    }
}
et-0/0/50 {
    mtu 9216;
    unit 0 {
        description facing_spine-03;
        family inet {
            mtu 9000;
            address 192.168.0.17/31;
        }
    }
}
et-0/0/51 {
    mtu 9216;
    unit 0 {
        description facing_spine-04;
        family inet {
            mtu 9000;
            address 192.168.0.21/31;
        }
    }
}
irb {
    mtu 9216;
    unit 1 {
        description LOCAL_SERVERS;
        family inet {
            mtu 9000;
            address 172.16.2.1/24;
        }
    }
    unit 100 {
        family inet {
            address 10.1.1.3/24;
        }
    }
}
lo0 {
    unit 0 {
        family inet {
            address 10.0.0.7/32;
        }
    }
}
vme {
```

```
        unit 0 {
            family inet {
                address 172.32.32.103/24;
            }
        }
    }
}
routing-options {
    router-id 10.0.0.7;
    autonomous-system 202;
    forwarding-table {
        export PFE-LB;
    }
}
protocols {
    igmp {
        interface xe-0/0/0.0;
    }
    ##
    ## Warning: requires 'bgp' license
    ##
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.44 {
                peer-as 100;
            }
            neighbor 192.168.0.28 {
                peer-as 101;
            }
            neighbor 192.168.0.16 {
                peer-as 102;
            }
            neighbor 192.168.0.20 {
                peer-as 103;
            }
        }
    }
    pim {
        rp {
            static {
                address 10.0.0.4;
            }
        }
        interface all;
    }
    lldp {
        interface all;
```

```
    }
    igmp-snooping {
        vlan all;
    }
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/28 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
                route-filter 172.16.0.0/21 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
    policy-statement bgp-clos-out {
        term loopback {
            from {
                protocol direct;
                route-filter 10.0.0.7/32 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term server-L3-gw {
            from {
                protocol direct;
                route-filter 172.16.2.1/24 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
    }
}
switch-options {
    vtep-source-interface lo0.0;
}
vlans {
    SERVER {
        vlan-id 1;
        l3-interface irb.1;
    }
```

```
    foobar {
        vlan-id 100;
        vxlan {
            vni 100;
            multicast-group 225.10.10.10;
        }
    }
}
```

LEAF-04

Here is the full Junos configuration of LEAF-04:

```
## Last commit: 2014-07-28 19:28:10 PDT by root
version "14.1-20140727_rt2_53_vjqfd.0 [dc-builder]";
/*
 * dhcpd-generated /var/etc/dhcpd.options.conf
 * Version: JDHCPD release 13.2X51-D20.2 built by builder on 2014-04-29 09:09:04
UTC
 * Written: Mon Jul 28 19:50:10 2014
 */
system {
    host-name temp-leaf-03;
    time-zone America/Los_Angeles;
    services {
        ssh {
            root-login allow;
            max-sessions-per-connection 32;
        }
        netconf {
            ssh;
        }
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
        file interactive-commands {
            interactive-commands any;
        }
    }
    extensions {
        providers {
            juniper {
                license-type juniper deployment-scope commercial;
            }
            chef {
                license-type juniper deployment-scope commercial;
            }
        }
    }
    processes {
        dhcp-service {
            traceoptions {
```

```
                file dhcp_logfile size 10m;
                level all;
                flag all;
            }
        }
        app-engine-virtual-machine-management-service {
            traceoptions {
                level notice;
                flag all;
            }
        }
    }
}
interfaces {
    interface-range ALL-SERVER {
        member xe-0/0/*;
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members SERVER;
                }
            }
        }
    }
    xe-0/0/0 {
        unit 0 {
            family ethernet-switching {
                interface-mode access;
                vlan {
                    members foobar;
                }
            }
        }
    }
    et-0/0/48 {
        mtu 9216;
        unit 0 {
            description facing_spine-01;
            family inet {
                mtu 9000;
                address 192.168.0.45/31;
            }
        }
    }
    et-0/0/49 {
        mtu 9216;
        unit 0 {
            description facing_spine-02;
            family inet {
                mtu 9000;
                address 192.168.0.29/31;
            }
        }
    }
    et-0/0/50 {
        mtu 9216;
        unit 0 {
            description facing_spine-03;
```

```
            family inet {
                mtu 9000;
                address 192.168.0.17/31;
            }
        }
    }
    et-0/0/51 {
        mtu 9216;
        unit 0 {
            description facing_spine-04;
            family inet {
                mtu 9000;
                address 192.168.0.21/31;
            }
        }
    }
    irb {
        mtu 9216;
        unit 1 {
            description LOCAL_SERVERS;
            family inet {
                mtu 9000;
                address 172.16.2.1/24;
            }
        }
        unit 100 {
            family inet {
                address 10.1.1.3/24;
            }
        }
    }
    lo0 {
        unit 0 {
            family inet {
                address 10.0.0.7/32;
            }
        }
    }
    vme {
        unit 0 {
            family inet {
                address 172.32.32.103/24;
            }
        }
    }
}
routing-options {
    router-id 10.0.0.7;
    autonomous-system 202;
    forwarding-table {
        export PFE-LB;
    }
}
protocols {
    igmp {
        interface xe-0/0/0.0;
    }
    ##
    ## Warning: requires 'bgp' license
```

```
    ##
    bgp {
        log-updown;
        import bgp-clos-in;
        export bgp-clos-out;
        graceful-restart;
        group CLOS {
            type external;
            mtu-discovery;
            bfd-liveness-detection {
                minimum-interval 350;
                multiplier 3;
                session-mode single-hop;
            }
            multipath multiple-as;
            neighbor 192.168.0.44 {
                peer-as 100;
            }
            neighbor 192.168.0.28 {
                peer-as 101;
            }
            neighbor 192.168.0.16 {
                peer-as 102;
            }
            neighbor 192.168.0.20 {
                peer-as 103;
            }
        }
    }
    pim {
        rp {
            static {
                address 10.0.0.4;
            }
        }
        interface all;
    }
    lldp {
        interface all;
    }
    igmp-snooping {
        vlan all;
    }
}
policy-options {
    policy-statement PFE-LB {
        then {
            load-balance per-packet;
        }
    }
    policy-statement bgp-clos-in {
        term loopbacks {
            from {
                route-filter 10.0.0.0/28 orlonger;
            }
            then accept;
        }
        term server-L3-gw {
            from {
```

```
                route-filter 172.16.0.0/21 orlonger;
            }
            then accept;
        }
        term reject {
            then reject;
        }
    }
    policy-statement bgp-clos-out {
        term loopback {
            from {
                protocol direct;
                route-filter 10.0.0.7/32 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
        term server-L3-gw {
            from {
                protocol direct;
                route-filter 172.16.2.1/24 orlonger;
            }
            then {
                next-hop self;
                accept;
            }
        }
    }
}
switch-options {
    vtep-source-interface lo0.0;
}
vlans {
    SERVER {
        vlan-id 1;
        l3-interface irb.1;
    }
    foobar {
        vlan-id 100;
        vxlan {
            vni 100;
            multicast-group 225.10.10.10;
        }
    }
}
```

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X

Z

About the Author

Douglas Richard Hanks Jr. is a Chief Data Center Architect with Juniper Networks and focuses on solution architecture for the data center. He works in the Switching, Security, and Solutions Unit (S3BU) that is responsible for the Juniper EX, QFX, and SRX Series hardware, software, and solutions.

Previously, he was a Solution Architect in the Routing Business Unit (RBU) with Juniper Networks supporting data center solutions with the Juniper MX platform. Prior to working in the business units, Douglas was a Senior Sales Engineer for Juniper Networks and supported large enterprise accounts such as Chevron, Hewlett-Packard, and Zynga.

Douglas is certified with Juniper Networks as JNCIE-ENT #213 and JNCIE-SP #875. Douglas' interests are network engineering and architecture for enterprise and service provider technologies. He is the author of the *Juniper MX Series* published by O'Reilly Media and several Day One books published by Juniper Networks Books.

Douglas is also the co-founder of the Bay Area Juniper Users Group (BAJUG). When he isn't busy with networking, Douglas enjoys computer programming and photography.

Douglas can be reached at *doug@juniper.net* or on Twitter @douglashanksjr.

About the Lead Technical Reviewer

Artur Makutunowicz has over five years of experience in Information Technology. He was a Technical Team Leader at a large Juniper Elite partner. His main areas of interests are service provider technologies, network device architecture and software-defined networking (SDN). He was awarded with JNCIE-ENT #297 certification. Artur was also a technical reviewer of the *Juniper MX Series* (O'Reilly Media) and "Day One: Scaling Beyond a Single Juniper SRX in the Data Center" (Juniper Networks Books). He is currently an independent contractor and can be reached at *artur@makutunowicz.net*.

About the Technical Reviewers

Many Junos engineers reviewed this book. They are, in the author's opinion, some of smartest and most capable networking people around. They include but are not limited to: Satish Surapaneni, Lakshmi Namboori, Salman Zahid, Sridhar Talari Rajagopal, Rakesh Dubey, Rahul Kasralikar, Apoorva Jindal, Masum Mir, Stephen Su, and Sathish Shenoy.

Special thanks to Kurt Bales and Jay Wilson, who helped provide technical content and reviewed the network automation and network analytics chapters.

Colophon

The animal on the cover of *Juniper QFX5100 Series* is a satin bowerbird (*Ptilonorhynchus violaceus*). The satin bowerbird is a medium-sized bird that measures, on average, one foot long and weighs half a pound. It is native to eastern and southeastern Australia, and also lives in the isolated wet tropics of northern Queensland. Many of these birds are rainforest inhabitants, especially around the Atherton Tablelands to the southwest of Cairns.

The appearance of males and females of the species is quite different. Females are greenish-brown with scalloped patterning; their bright lilac eyes stand out against this background color. The black plumage of adult males often looks blue and metallic, and their bills are bluish-white. Early in life, males closely resemble females in coloring, but they attain adult plumage between their fifth and seventh years.

Satin bowerbirds are largely frugivorous, yet they will consume a varied diet. Insects are the typical food in the summer, and leaves make up the bulk of their diet in the winter. Satin bowerbirds can be a nuisance for farmers since they often raid fruit and vegetable crops.

The birds' namesake bowers are built of twigs and leaves, and adorned with shiny objects (sometimes these things are even stolen from other bowerbirds) or painted with berries and charcoal. However, bowers are not nests. Built on forest floors, the males build bowers as bachelor pads where they attempt to woo females after attracting their attention with an ornate show of calls and strutting. On average, young bowerbirds only attract female visitors less than 10% of the time, and many females don't deign to actually enter their bowers. Success rates rise steadily as males age.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from Cassell's *Natural History*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.