

Chapter 1

Getting Familiar with JUNOS Software

In This Chapter

- ▶ Understanding the purpose of a network operating system
 - ▶ Discovering how JUNOS software is different
 - ▶ Looking inside JUNOS software
 - ▶ Realizing why the JUNOS development process matters
-

Have you ever considered what it takes for the operating system (OS) on your computer to keep everything running smoothly? How it must allocate system resources and manage tasks so that you can keep typing, even while you're printing or an application is performing an autosave? If you type for most of the day, you probably notice it a lot.

Now, consider everything that a network operating system (NOS) must do to keep your network running smoothly. Not only does it need to process a gazillion details, such as figuring out how to direct traffic from each point A to each point Z, but it also has the weighty task of delivering the bazillions of packets cruising your network to their destinations. And, don't forget the NOS has to track and apply all those special services and security rules that you've requested.

The engineers of a network operating system have their work cut out for them, not only because of all the different design and development options, but because everyone wants their network to be always ready, and fast, without pauses, such as a momentary freezing of the cursor. Just as the OS that you use on your desktop affects the way that you do certain tasks and the efficiency of your computer, the operating system that runs your networking platforms impacts their speed and ease of management, not to mention your network's reliability and security. In this chapter, we explore JUNOS software.

Exploring the Functions of Network Operating Systems

Network operating system (NOS) describes a broad category of software—from basic systems that support only the functions of local connections, such as printer sharing and file system, to sophisticated systems running on platforms specifically designed as networking infrastructure. Our focus is on the latter—the software systems that support not only local, but also metro and wide area networking infrastructure, including the networking platforms of the worldwide Internet.

It's all about control

In building your network, you use various types of specialized devices that pass along traffic from one to another. We speak of *internetworks* when we interconnect these individual networks. Before a network can safely deliver traffic, each device node must first know where to send each bundle of traffic, or *packet*, that enters through its interfaces.

This essential map for connectivity and other orchestrating processes are the functions of the network operating system's *control plane*.



In simplest terms, the *control plane* is the brain of the networking device with the *forwarding plane* providing the brawn to quickly move packets through the system.

The processes and information of the control plane must provide answers to two essential questions:

- ✓ How does the network direct the delivery of packets from one place to another? In other words, what are the routes or paths to establish, how do they change, and how does each device know which to use for each packet?
- ✓ What does the network do with each of the packets along its journey? In other words, what are the handling rules, or *policies*, established for traffic delivery?

While the questions can be simply stated, the possible responses are virtually limitless. You can define dozens and dozens of protocols to answer these questions for different types of network maps and all the different types of traffic, not to mention how the control plane monitors and manages it all. The many processes to control the network delivery of packets fills the industry with all those three- and four-letter acronyms that you've somehow managed to file into your memory (or if you're like us, you've just filed away the place to look them up when you need them).

Moving on: Packet forwarding

Along with assembling the intelligence to properly deliver the traffic from one place to another, the network operating system (and its hardware) must actually deliver packets to the correct destination using this intelligence. Moving packets through a networking device is the function of the network operating system's *forwarding plane*, also sometimes known as the *data plane*.

Packet forwarding takes care of the needed handling to move each packet quickly from its inbound device interface to the proper outbound interface(s). For large networking devices that carry terabits of traffic, this handling must occur at an ultra-über fast rate to maintain the high packet throughput of the machines.

The sophisticated service demands of today's users and applications have further added to these responsibilities. While forwarding the packets, the networking devices must also typically apply a range of filters, policies, and services for protecting the network (and its clients and applications) and assigning priorities for the use of its resources. Visualize watching a YouTube video. Now visualize all your users watching the video all at the same time because some clown in your office passed along an e-mail with the link to everyone. And now think about all the traffic hitting your network all at once, just as your president is on a critical call with your biggest customer. Oofta! It's just one example of where you may want to define a few of those extra rules that your network can follow in making its packet deliveries. (We provide Chapter 15 to help you out.)

Taking Advantage of One Network OS

Network operating systems have a lot to do and can have a big impact on the performance, ease of operations, reliability, and security of your network. JUNOS software is different, in that it's one operating system enhanced along one software release train built in one modular architecture. But, why does having one operating system matter?

One operating system means the Juniper engineers build upon the same, single source code base and then share this code, as appropriate, across all the platforms running JUNOS software. For example, enterprise platforms use the same hardened implementation of the routing protocol Open Shortest Path First (OSPF) that has been running in large service provider networks for many years. It's not a different code set, but the same one. (To set up OSPF, see Chapter 8.)

So, if your responsibilities include administrating the network, you find that many features are configured and managed in the same way on the different platforms. One operating system therefore saves you time, potentially lots of

20/20 through the rear view

Most of the engineers writing the early releases of JUNOS software came from other companies where they had previously built network software. They had first-hand knowledge of what worked well, and what could be improved because they'd experienced the issues that occur when they didn't have good control over the code development. The software became difficult to manage, and changes could result in unpredictable problems that often became visible only in the operational networks of customers.

So, in engineering the principles of development and architecture for JUNOS software, these

engineers found new ways to solve the limitations that they'd experienced in building the older operating systems (the ones that still exist today in many networks, surviving through lots of different versions, patches, and administrator workarounds). These innovations in JUNOS software are rooted in its earliest design stages. They can't be retrofitted or reverse-engineered into existing systems. The founding engineering team built them into the origins of JUNOS software, and they have passionately preserved them over the years.

it, in everything from training to setup to ongoing operations. And, if you plan changes in the network, one operating system can save you time there, too. With far less variation to evaluate, test, and deploy, it's less effort for feature roll out, software upgrades, and other network modifications.

Taking a Peek Inside JUNOS Software

Another area impacting the reliability, performance, security, and many aspects of your devices (and therefore your network) is the way that the engineers architect the network operating system — in other words, how they organize its many processes and how these processes work together.

Fundamental to the architecture design of JUNOS software are its modularity and Juniper's innovation to separate the functions of control from packet forwarding. (For more on packet forwarding, see the section "Moving On: Packet forwarding," earlier in this chapter.)

Going their separate ways: Functions

In JUNOS software, the functions of control and forwarding are cleanly divided. The explicit division of responsibility allows the software to run on two different engines of processing, memory, and other resources. Separation of the control and forwarding planes is a key reason why JUNOS software can support so many different types of hardware platforms from a single code base.

Figure 1-1 provides a high-level view of the JUNOS software architecture with its two functional processing planes. Shown above the dashed line is the control plane that runs on what is known as the *Routing Engine (RE)* of the Juniper device. Below the dashed line is the packet forwarding plane, which usually runs on a separate *Packet Forwarding Engine (PFE)* of the device, along with switching boards and services cards in larger Juniper platforms.

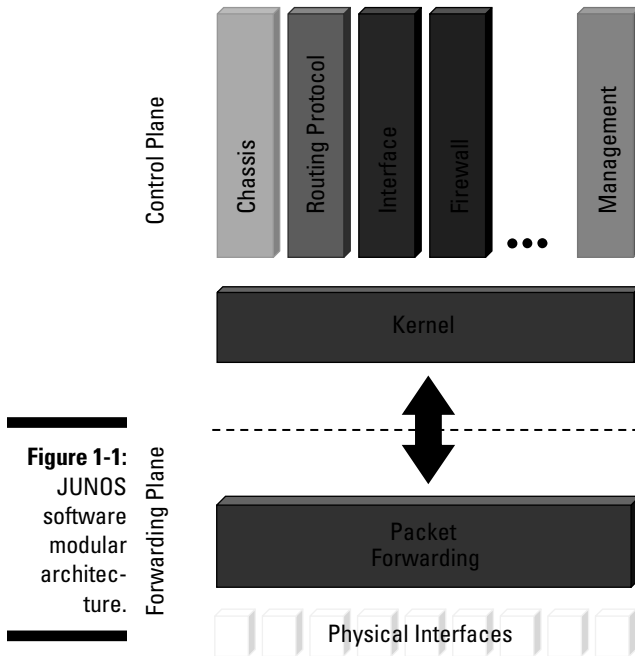


Figure 1-1:
JUNOS
software
modular
architec-
ture.

The fundamental architecture decision to separate control from forwarding provides important design benefits in scalability and reliability.

Do you want faster platforms in your network? That's like asking if you'd like to have today off (with pay, of course). Yes, it would be good to have the network go a little faster. Faster, faster, faster is a constant drum beat for networks. And, in its ten years of product delivery, Juniper has scaled the throughput of its fastest devices from 40GB per second to multiple terabits per second. The use of separate processors for the RE and the PFE has been the essential architecture element to each performance breakthrough. In particular, separation lets the PFE throughput scale with the increasing speeds of the custom *application-specific integrated circuits (ASICs)* on which it runs.

Separating the engines also reduces interdependencies between them. Not only does this separation help preserve the operation of each when the other is experiencing problems, it also gives the Juniper engineers more ways to

provide system redundancy and failover. For example, you find dual REs in some platforms, while the EX 4200 Ethernet switches offer a capability called Virtual Chassis to provide redundancy, among other benefits. (See Chapter 3 for the details of this switching feature).

Plain smart: The planes of JUNOS software

All the necessary functions of the control plane run on the Routing Engine (whether you have a router, switch, or security platform running JUNOS software). Figure 1-1, shown earlier in this chapter, shows the high-level design of the control plane — a set of modules, with clean interfaces between them, and an underlying kernel that controls the modules and manages all the needed communication back and forth between all the components. The kernel also handles the RE communications with the Packet Forwarding Engine. Each of the different modules provides a different control process, such as control for the chassis components, Ethernet switching, routing protocols, interfaces, management, and so on.



The basis of the JUNOS kernel comes from the FreeBSD UNIX operating system, an open source software system. This mature, general-purpose system provides many of the essential basic functions of an operating system, such as the scheduling of resources. To transform it into a network operating system, the Juniper engineers extensively modified and hardened the code for the specialized requirements of networking.

You may be wondering if you have a way in JUNOS software to protect the control plane itself from a security attack. Yes, you can configure filters and rate limit the traffic that reaches your RE. (For more on this topic, go to Chapter 9).

The Packet Forwarding Engine is the central processing element of the forwarding plane, systematically moving the packets in and out of the device. In JUNOS software, the PFE has a locally stored *forwarding table*. The forwarding table is a synchronized copy of all the information from the RE that the forwarding plane needs to handle each packet, including outgoing interfaces, addresses, and so on. Storing a local copy of this information allows the PFE to get its job done without going to the control plane every time that it needs to process a packet.

Another benefit to having a local copy? The PFE can continue forwarding packets, even when a disruption occurs to the control plane, such as when a routing or other process issue happens. (Find out more about this capability and what other features are important to preserving high uptime in the operating system in Chapter 5.)

That's not a problem: The many benefits of modular architecture

Have you ever had a router continually reboot, and when you look on the console, you see that an error occurred in a single nonessential process?

With JUNOS software, you don't see that problem. The modular architecture of JUNOS software allows individual control plane processes to run in their own module (also sometimes called a daemon). Each module has specified processing resources and runs in its own protected memory space, avoiding the processing conflicts that can occur.

What about a minor hiccup in SNMP bringing down your whole system? That's another misfortune that you won't miss with JUNOS software, because its clean separation between control processes helps to isolate small problems so that they can't create worse havoc. If a malfunction in a module causes an issue, the rest of the system can continue to function. For example, one module can't disrupt another by scribbling on its memory.

In our many discussions with users, we hear over and over again that the stability of JUNOS software is the biggest difference that they see after deploying Juniper platforms in their network. They tell us about their boxes running for months and months, even years without interruption. How they popped it in the rack, set up the configurations, and never looked back. It just keeps going and going and going.

Want to know another benefit of the modular architecture? It eases fault isolation. With each process functioning within its own module, when the occasional problem does occur, pinpointing the exact reason is far less complicated for both you and the Juniper support team. With quick identification and a good understanding of the root cause, you can apply a fix that works, from the first time.

We have one more benefit to highlight — flexible innovation. The organized structure of the JUNOS software architecture enables deep integration of new capabilities with highly functional interaction with existing processes. What does it mean for you? Native support of new services and features delivers a richness of capability with the high performance that you expect. For example, among recent integrations to JUNOS software are the security services being derived from Juniper's ScreenOS operating system.

Developing JUNOS Software

Software development probably isn't a topic that you expected to find in a networking book. After all, you don't need to build JUNOS software. Juniper's engineers already do that for you.

However, we include this topic because we think it's important. Software development is an essential reason why JUNOS software is different and why organizations see such a difference in the stability of their networking devices when they put JUNOS software into their infrastructure. When a development process produces code that is consistently more stable and predictable, all aspects of operations just get a lot easier.

You may be wondering how Juniper engineers can use only a single base of the source code, given all the different features and platforms supported by JUNOS software. (We discuss the single source code base in this chapter in the section "Taking Advantage of One Network OS.")

It does take a well-defined process with lots of discipline to stick to the principles. The answer begins by following a single software release train for development. The Juniper engineers extend the one source code base of JUNOS software in highly scheduled, incremental steps, where everyone adds their code (and any needed fixes) to that same repository. The expanding repository forms the main line of code, released in a series of numbered versions, as shown in Figure 1-2.

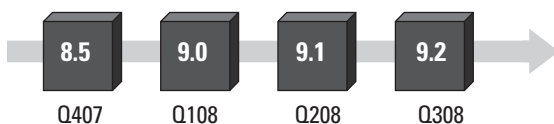
With all those versions, you may be wondering about the answers to a few questions:

- ✔ **How can you find out whether a feature that you're using is also available in a new version that you want to deploy?** Well, you don't have to worry about that. If it's in your current JUNOS software release, it's generally in the new version. By following a main line, the Juniper engineers deliver supersets of features with each release.
- ✔ **How will you choose which release to use?** With other systems, you may be used to having to make decisions about tradeoffs. One software train has all the features you need, but doesn't support the latest hardware; this other train has all the features you want, but has a critical bug in one critical feature you need; and yet another train has all the critical features, but lacks two less important features you have used.

With JUNOS software you don't have these trade-offs. For each new version, the Juniper engineers compile the full set of release binaries so that each version is available to run all the Juniper platforms run by JUNOS software. Each binary brings together the features from the repository that are relevant (and tested) for the particular platform that it runs.

✔ **What about when you report a bug?** How will you know whether it's fixed on every platform? Because all platforms share much of the core code, most bug fixes automatically get integrated into all the devices where you find that feature. So if it's fixed in one platform, it's fixed in all (unless it's minor and missed the test window for that platform, but even then, it won't be too far behind).

Figure 1-2:
JUNOS single software release train.



Reaping the benefits of a single release train

A single software release train is extremely rare, not only for networking equipment, but for any software development. The more common approach, particularly when rapid changes are needed, is to branch code — in other words, to create a new release train by taking a copy or a subset of the existing code. Then two main lines exist in which the developers work and add new features. And sometimes, for any given feature, two different developers might create the code for the same feature. And when the pressure mounts for rapid changes, the code might be copied again, starting up yet another

Testing, testing

Juniper's testing processes are extensive and highly automated. With only the one code base to worry about, they can really put it through the ringer. Ask someone from Juniper about it sometime. All these development disciplines are there to deliver highly stable code to you.

Automated regression testing takes days to run in Juniper's high-speed test beds. The regressions tests check that previously released

features still work as expected, while new test scripts check the new features of the release. Here the single release train discipline aids Juniper's engineers because they can focus on just one set of code and not have to worry about many, many different trains, or branches, or patches of code. It's one of the secrets of how they keep on track, delivering new versions four times a year.

Asking the right questions

Consider reviewing the software development processes of vendors as a part of your evaluation process for new network and security equipment, because it can save you time and money down the road. Here are some questions that you can ask your vendors about their software development process:

- ✓ How many different software versions exist for the products you are buying? How do you know when to use one version versus another?
- ✓ What steps do development engineers follow when adding new feature code? How do they support different software versions or release trains? How do they decide which features to add to which version?
- ✓ What are the steps for adding fixes to the code? What procedures ensure that a new fix is a part of all releases, including those in the future?
- ✓ How are newly developed features (and fixes) tested? What guidelines determine when a release is ready for customers? Can a new release affect previously working features?

train. Over time, these tactical decisions can end up in hundreds and even thousands of different strains of the software. And that's within what is supposedly the same operating system. We haven't even discussed the situation where a vendor might create a whole new operating system for each new platform.

So what does it all mean to you? The bottom line is that you get lots of new features in a series of stable releases.

When deploying JUNOS software for the first time in their networks, administrators tell us that they immediately notice its stability and reliability. We hear about big drops (20, 40, 50 percent, sometimes even higher) in the total number and the duration of their unplanned system events. More stable code just makes their jobs a lot easier because it reduces their risk of business disruption and the frequency of their unplanned maintenance, as well as streamlining upgrade procedures when they do want to make changes to the version running their machines.

Hup 1, 2, 3, 4: JUNOS software release numbering

Essential to the success of the single software release train of JUNOS is the frequent release of new versions. Otherwise, you and other users may become impatient waiting for new functionality. Each new major release of JUNOS software is called a new version. Juniper numbers its new versions to the first decimal point, as in the 9.2 JUNOS software release. You sometimes see jumps in

the release numbering to the next whole number. It used to mean the delivery of extensive new functionality, but now frankly, it's a more arbitrary decision.

Juniper also issues maintenance and service releases to fix issues found in a version after its first release. *Maintenance releases* are standard releases provided to all customers. Juniper delivers *service releases* between maintenance releases when it needs to address a specific issue for a specific customer.

The easiest way to understand the release name is to look at an example:

```
jinstall-9.2R2.10-domestic-signed.tgz
```

- ✔ The descriptor `jinstall` indicates the binary of the release; in this case, the image is for the routing platforms.
- ✔ Next, is the release version: 9.2.
- ✔ Following, is a letter that indicates the type of release. As a customer, you generally see R for released software (another type is B for Beta releases).
- ✔ Next appears the maintenance release number (2 in our example).
- ✔ The last number is the spin of that maintenance release, in this case 10. The Juniper engineers create different spins for different testing versions of the software and also to create the service releases.
- ✔ Finally, this name indicates that the software is for domestic use (in the United States and Canada). The other type is worldwide. The difference is primarily their inclusion of encryption capabilities.

