JUNIPER NETWORKS | Engineering Simplicity

DAY ONE: DEPLOYING BGP RIB SHARDING AND UPDATE THREADING

By Ravindran Thangarajah

DAY ONE: DEPLOYING BGP BIB SHARDING AND UPDATE THREADING

BGP RIB Sharding (BRS) and Update Threading (UT) are aimed at scaling up BGP convergence performance by taking advantage of the multicore (multiprocessing) capabilities provided by modern processors. By using the paradigm of parallel software execution on a multiprocessing system, these features speed up BGP processing and help deliver superior convergence performance. This book starts with a look at the design approach for these features, then the various factors that need to be considered for best performance . It reviews targeted customer use cases and their configuration aspects and operational commands. It's the perfect *Day One* book, short, precise, and it can get you up and sharding on day one.

"Scaling up with better convergence time is a mandatory requirement for many large organizations to which BGP RIB Sharding is one of the optimal solutions. This is a power-packed book which explains the need, basics, and configuration steps of BGP RIB sharding. The author conveniently explains concepts and the book has good flow of topics covering various use cases. I would highly recommend this book for those engineers looking forward to scaling up their environment and who want to leverage Juniper's multicore capable systems."

Nupur Kanoi, Sr. Engineer, Juniper Networks Ambassador
3xJNCIE (JNCIE-SP, JNCIE-DC, JNCIE-ENT)

"Every system, whether it's a database, a physical system (for example a switch or router), or in this case a routing protocol, at one point they all face the limits of scaling up. Distributed systems, scale out, and sharding are the next steps in scaling. This book describes in high detail how BGP Sharding works and how to configure the Juniper BGP implementation to take advantage of multiprocessing capabilities present in modern CPUs so that BGP pipeline processing can happen in parallel. You can learn a lot by reading these pages."

Melchior Aelmans, Lead Engineer Cloud Providers, Juniper Networks

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

- Work with the various factors that should be considered before deploying BGP RIB Sharding and Update Threading.
- Target use cases that can benefit when deploying various BGP Sharding features.
- Determine what feature sets will be of benefit for your deployment.
- Configuration the correct parameters to use for deriving best performance.
- Resource deployments on third-party servers.

ISBN 978-1941441985
52000
9 781941 441985

DAY ONE: DEPLOYING BGP RIB SHARDING AND UPDATE THREADING

Thangarajah

Day One: Deploying BGP RIB Sharding and Update Threading

By Ravindran Thangarajah

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Ravindran Thangarajah
Technical Reviewers: Ashutosh Grewal, Sanjay Khanna, Ashish Kumar, Vinay Kumar, Jaihari Loganathan, Balaji Rajagopalan, Jay Ramalingam
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel

Printed in the USA by Vervante Corporation.
Version History: v1, January. 2021
2 3 4 5 6 7 8 9 10

Send errata and comments to dayone@juniper.net

About the Author

Ravi Thangarajah is a Distinguished Engineer at Juniper Networks based out of Bengaluru, India. He has over 25 years of experience in the networking industry building products spanning the range from simple ATM/Ethernet switches to complex multi-chassis IP/MPLS routers. He enjoys collaborating with customers and solving problems for them. He has a Masters degree in Computer Science. His areas of interest include distributed systems, system analysis and troubleshooting, system performance, and software re-engineering.

Author's Acknowledgments

Writing this book was a process of carefully cataloging the numerous considerations that come into play when software performance is a key goal. It was also an opportunity for me to showcase the excellent work that the team has done and its impact. I would like to thank the team of developers and testers whose work I had the privilege of highlighting here.

I would like to thank Jay Ramalingam, my manager, who was instrumental in driving the need for this book. I would like to thank all the reviewers for their thorough reviews and for helping me identify the topics and their flow. Special thanks to Kedar Kawadkar for performing all the tests, collating the results, and providing me with all the required performance data. Many thanks to our editor-in-chief Patrick Ames, for guiding me through the whole process of writing this book.

I would also like to thank my wife Subashree, daughter Keerthana, and son Krithik, for their help and support at home.

Welcome to Day One

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

- Download a free PDF edition at http://www.juniper.net/dayone.
- PDF books are available on the Juniper app: Junos Genius.
- Purchase the paper edition at Vervante Corporation (www.vervante.com) for between $15-$40, depending on page length.

Key BRS/UT Resources

This *Day One* book distills the knowledge and information that is required to derive maximum benefit from these outstanding BGP performance features. Scripts and tools to aid deployment of BRS/UT will be available soon on the author's GitHub repository: https://github.com/juniper/bgp-sharding.

The Juniper TechLibrary supports BGP with its excellent documentation. This book is not a substitute for that body of work. Take the time to review the documentation here: https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/config-guide-routing/config-guide-routing-bgp.html.

What You Need to Know Before Reading This Book

- It is assumed that the reader is familiar with basic workings of a network and routing device. The reader is also expected to have a basic understanding of the BGP protocol.
- It is assumed that the reader is familiar with the basics of the Junos OS and has at least a high-level exposure to Juniper platforms. A high-level understanding of BGP-related configuration and operational commands in Junos would be of help as well. The Day One library has several books on Junos at http://www.juniper.net/dayone.

What You Will Learn by Reading This Book

- Work with the various factors that should be considered before deploying BGP RIB Sharding and Update Threading.
- Target use cases that can benefit when deploying various BGP Sharding features.
- Determine what feature sets will be of benefit for your deployment.
- Configure the correct parameters to use for deriving best performance.
- Resource deployments on third-party servers.

About This Book

The Border Gateway Protocol (BGP) is one of the most widely deployed protocols in networks around the world. BGP is a versatile protocol that can be used within a network and between networks to exchange a variety of network reachability information spanning multiple layers (such as transport, routing, and service) of a network. Due to the way it is used, BGP typically handles very high-route scaling, sometimes running into the tens of millions of routes. The high-route scale, combined with the centrality of BGP to modern network operations, makes BGP convergence performance a critical requirement for today's networks. The faster the convergence performance during critical network events, the lower the time the network is potentially down and/or used sub-optimally. This in turn makes for higher levels of service availability for an operator's end customers.

BGP RIB Sharding (BRS) and Update Threading (UT) are aimed at scaling up BGP convergence performance by taking advantage of the multicore (multiprocessing) capabilities provided by modern processors. By using the paradigm of parallel software execution on a multiprocessing system, these features speed up BGP processing and help deliver superior convergence performance. As we show later, deploying these features in the right use case scenarios after carefully considering all deployment considerations can make convergence performance in certain scenarios up to ten times faster!

This book starts with a look at the design approach for these features in Chapter 1. Chapter 2 discusses the various factors that need to be considered for a deployment decision and for getting the best performance out of these features. Chapter 3 goes over some targeted customer use cases and discusses performance results for them. Configuration aspects and operational commands for these features are covered briefly in Chapter 4, and the book concludes with some final thoughts and highlights of enhancements planned for the future in Chapter 5.

Glossary of Terms

- BRS – BGP RIB Sharding: RPD feature for parallelizing BGP pipeline processing.
- cRPD – Containerized RPD: Virtualized RPD capable of running in standard container environments.
- RPD – Routing Protocols Daemon: Daemon in Junos OS that implements most of the control plane protocols (Layer 3).
- UT – Update-Threading: RPD feature for parallelizing BGP update construction and transmission.
- vRR – Virtual Router Reflector: Juniper's virtualized Route Reflector product.

Chapter 1

Introducing Sharding and Update Threading

First let's take a look at the design of BGP RIB Sharding (BRS) and Update Threading (UT). You'll get familiar with the BGP protocol pipeline operations and how these features help improve its performance in Junos.

BGP Pipeline

BGP protocol processing can be viewed as a pipeline of operations that operate on information encapsulated in BGP Update messages. Information in the Update message includes destination prefixes and their attributes. This pipeline (Figure 1.1) enables a BGP node to learn routes from Update messages for its use and to advertise routes to BGP peers.

Figure 1.1 *BGP Pipeline*

The following operations happen in the BGP pipeline shown in Figure 1.1:

- The BGP Update is received from a BGP peer.
- The Update is parsed and information in it is extracted.

- Input policies, if any, are applied. Route entries are created in the Routing Information Base (RIB) for all destination prefixes in the Update message.
- The route is resolved, if needed, to identify the next hop to be used for that route. Best path selection is done to identify the best route.
- Output policies are applied on RIB routes. Those routes that need to be advertised are converted into an intermediate form.
- The routes to be advertised are coalesced and formatted into BGP Update messages.
- Update messages are written to TCP sockets and sent to BGP peers.

RIB Sharding

The key idea behind BGP RIB Sharding (BRS) is to take advantage of multiprocessing capabilities present in modern CPUs so that BGP pipeline processing can happen in parallel as shown in Figure 1.2.

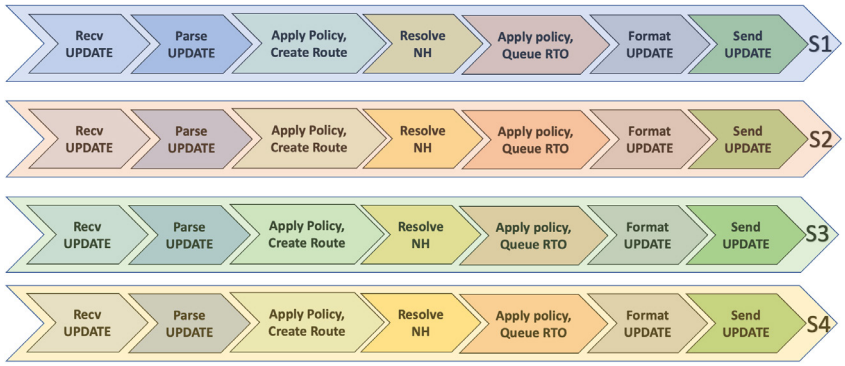

Figure 1.2 *Running BGP Pipeline in Parallel*

In Figure 1.2 you can see the four instances of the pipeline (S1 through S4) potentially running in different cores (processors) in a CPU. Each instance is a unique thread of execution within the RPD process and is referred to as a *shard thread*. In order for shard threads to run in parallel with minimal need to coordinate them with other threads, the system:

- Partitions the BGP RIB into slices called shards. All prefixes that fall into a given shard are owned and handled by a unique shard thread. As Update messages reach the input side of the pipeline, the prefixes in it are examined and given to the shard thread that owns them for processing.

- Provides each shard thread with its own copy of the necessary state needed for running the pipeline.

Keeping the threads maximally independent is a key design attribute that helps BRS scale the system performance up by scaling the number of processors (cores) up, too.

Update Threading

One effect of the parallel execution model on the output side is that more Update messages might be generated when advertising to a downstream peer. This is because each shard thread generates its Update messages independently. While a single-threaded BGP pipeline could produce efficiently packed Update messages, multiple pipelines executing in parallel might produce more Update messages that are less efficiently packed. Since efficient packing of Update messages is a key requirement for the overall performance of the network, *Update Threading* (UT) was introduced into the system.

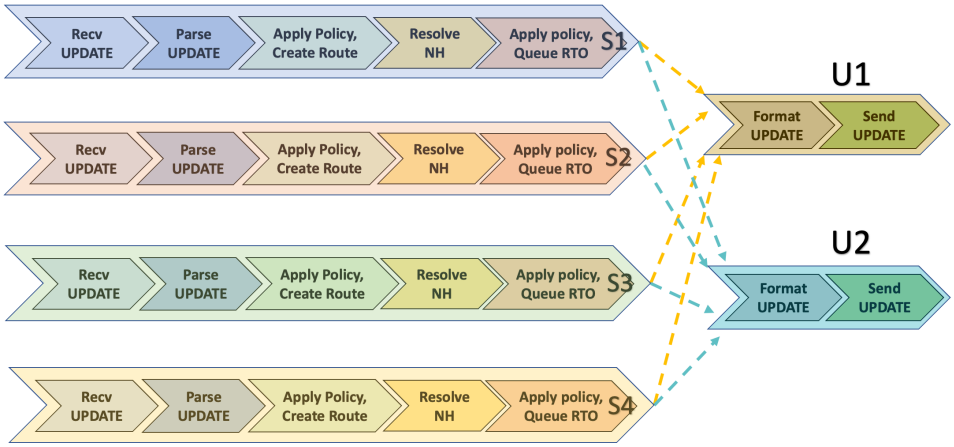

Figure 1.3 *Update IO Threads for Output Processing*

Update-IO threads (U1 and U2) execute the last output stages of the BGP pipeline. They receive the routes that need to be advertised along with their attributes from shard threads. Then they coalesce the received information into efficiently packed BGP Update messages and send them out to the downstream BGP peers.

UT could potentially be used even without BRS when there is a need to improve the output side performance alone. Note that when BRS is configured UT is required to be configured also.

Chapter 2

Deployment Considerations

This chapter discusses important factors that should be considered before deploying BRS and UT features. Understanding these factors helps you determine if you will benefit from these features and also enables you to get the best performance from them.

Is BRS/UT for Me?

If you do not have much time for details, Figure 2.1 is a quick flowchart with a set of steps that you can use to determine if there is a case for deploying BRS/UT. All the factors mentioned here are discussed in detail later in this chapter.

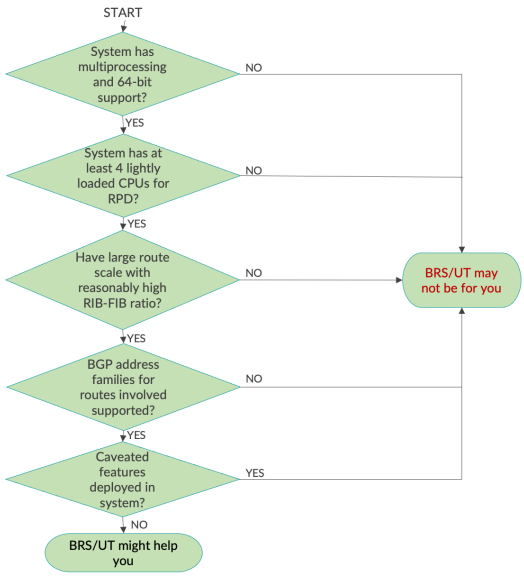

Figure 2.1 *Deployment Flowchart*

Now let's see if BRS/UT is for you by drilling down into all the considerations before making a final decision.

CPU Capabilities

Multicore/Multiprocessor 64-bit CPU

Scaling up software performance with parallel processing in a multiprocessor (multicore) system is the basis of BRS and UT. Typically, such systems use multicore CPU(s) – each CPU chip has multiple CPU cores embedded in it. Ensure that the system where the BRS/UT are being deployed is multiprocessing capable.

Some third-party systems use Hyper Threading (HT) as a means to provide multiple logical CPUs for software. Multiprocessing performance on such systems depends a lot on the workload. For BGP processing, we have observed that HT does not work well, and we recommend that it be turned off.

BRS/UT can only be enabled on systems that run the Junos Routing Protocols Daemon (RPD) in 64-bit mode. Note that the system used should support 64-bit RPD.

BRS/UT can be enabled on different types of Juniper products, and we cover how to check for multiprocessing capability in them next.

PTX/MX Routers

RPD runs in the Routing Engine (RE) of these routers, so ensure that the RE is multiprocessing capable and supports 64-bit RPD. Consult the router and RE hardware specification for information on these capabilities by following the links within the Juniper TechLibrary: https://www.juniper.net/documentation/.

TIP You can obtain router and RE information for a given system by using the `show chassis hardware` CLI command in Junos.

Here's a list of some popular REs that support multiprocessing and 64-bit capabilities on these platforms:

- RE-S-1800x4 and variants
- RE-S-X6-64G
- REMX2K-X8-64G
- RE-PTX-X8-64G
- RE-S-1600x8

JRR

JRR system is multiprocessing capable and supports 64-bit RPD.

vRR

vRR is deployed on third-party server platforms. The capabilities of the server platform need to be checked.

TIP On Linux servers use one or more of the following commands to check for multiprocessing and 64-bit support capabilities: `cat /proc/cpuinfo`, `dmidecode`, and `lscpu`. Consult the man page for details on these commands. Availability of some of these commands depend on the Linux distribution and the packages installed.

cRPD

cRPD is deployed using supported container environments on third-party server platforms. As with vRR, you need to check the capabilities of the server.

Available CPUs

Even in a multiprocessing capable system, there should be enough logical CPUs available to realize maximum possible performance gains. This is especially relevant for RPD running on a third-party server platform. An adequate amount of CPU resources should always be made available for maximizing parallel execution of shard and update-IO threads.

In a third-party server deployment, if you have a large pool of CPUs that can be made available for Junos/RPD, one—albeit liberal—option would be to ensure that the number of allocated CPUs is at least:

```
(No. of Shard Threads + No. of Update-IO Threads + M + N)
```

M (can be 1 or 2) is for the compute needed by other threads in RPD and N is for the compute needed for other critical processing in Junos outside of RPD. This scheme allows for maximum parallel execution of threads when the need arises. If CPUs are limited, then one can choose a CPU count that is as close to this option as possible.

The topic of choosing the right number of Shard and Update-IO threads is covered in detail in a subsequent section of this chapter.

CAUTION In some Juniper systems, not all of the CPUs present are made available for Junos execution. Some CPUs might be reserved for use by other software entities. The number of CPUs reserved for Junos should be considered in such systems.

CAUTION If enough CPU resources are not available, the performance of the system could dip below the level of baseline performance when these features are not deployed.

NOTE For third-party platforms we recommend lab testing of convergence scenarios with different numbers of logical CPUs to arrive at the appropriate number of CPU resources that should be reserved for Junos/RPD.

CPU Load

After ensuring multiprocessing and 64-bit capabilities, you should also pay attention to the processing load on the CPU. A system that is loaded heavily at steady state will not be able to provide the maximum benefits from these features. A heavily loaded system will have fewer free CPU cycles available for shard and update-IO threads to utilize when compared to a lightly loaded system. This will limit the performance gains that can be realized. The notion of what is considered as heavy load varies, but the higher the average steady state CPU load, the lower the amount of performance gains that are possible.

TIP On a Juniper system the `show system processes extensive` Junos CLI command provides information on CPU load. One can also use the `top` UNIX command from a UNIX shell to get CPU load information.

Input/Output (I/O)

BRS/UT performs best when shard and update-IO threads run and perform BGP pipeline operations without being made to wait for work. Even if a CPU is available, shard threads cannot make efficient use of it if BGP input PDUs are not coming to them at a rate that keeps them busy. Similarly, update-IO threads cannot send out BGP update PDUs at the maximum rate possible if they are throttled due to slow I/O.

There are two important factors to consider here. One is whether the deployment platform supports enough packet I/O bandwidth, and the other is whether the BGP peer on the other side of the BGP session is fast enough.

Platform

Juniper platforms are designed to provide adequate packet I/O bandwidth for efficient and high-performance protocol processing. While some extreme scenarios might strain the packet I/O capabilities in a Juniper platform, this is an important factor mainly for third-party servers running RPD. These servers should be able to support enough packet I/O bandwidth to allow for the best performance. Attention has to be paid to all factors that affect server I/O performance:

- Ensure that hardware virtualization technologies that improve I/O performance like SRIOV are enabled.
- Ensure that the server OS is up-to-date, has all the I/O performance patches needed, and drivers that support available hardware virtualization mechanisms are installed.
- Ensure that the server CPUs are configured to operate in high performance mode. Modern servers provide the ability to configure the CPUs at various performance levels for power consumption and more. For the best performance with BRS/UT, the CPUs should be configured in performance mode.

Slow Peers

Slow BGP peers can limit the performance gains from BRS/UT. Peer slowness can occur for a variety of reasons:

- Peer platform capabilities (slow hardware/software).
- Issues in the network connecting to the peer. This can be due to various underlying reasons including network congestion, network element issues, etc.
- TCP performance issues caused by a variety of factors.

Systematic step-by-step debugging and analysis is usually needed to root out the cause of slow peers. Ensure that the root cause is addressed so that you get the best performance out of BRS/UT.

Identifying I/O Issues

The simplest way to check for I/O issues would be to check the CPU utilization for RPD threads. During times of heavy BGP processing load, if you find that BRS/UT threads are not utilizing CPU fully, even when CPU capacity is available, it is likely that you are seeing some form of I/O bottleneck.

The UNIX `top` utility is handy for looking at CPU utilization details. There are command line options to this tool to get information on a per-thread basis. Use this option to look at the CPU utilization for shard and update threads. If this is not 100% when you expect it to be 100% (and the CPUs in the system are not fully utilized) then you are likely seeing an I/O issue.

Slow peers can be identified by looking at the TCP send/receive window size details for the TCP connection connecting to the peer. You can use the `tcpdump` utility to see what is happening on the TCP connection and check window sizes. More detailed analysis can be done using tools like `Wireshark` (see: www.wireshark.org).

You can use the Junos CLI command `show bgp neighbor statistics` to check for I/O issues on the output side (when advertising to downstream peers). A non-zero value

for "Write Queue Blocks" indicates blocks as it attempts to send BGP messages to the corresponding peer. When UT is already deployed you should use the `show bgp neighbor statistics update-io <update thread name>` version of this command to get write queue blocks. This version also shows "Peer designated a slow writer" information that tells you whether the peer has been designated as slow.

RIB-FIB Ratio

Benefits from BRS/UT manifest only when there is a certain amount of route scale in a system. At lower route scales the overhead of multithreading will begin to dominate, and the system can perform poorly when compared with the non-multithreaded mode. Even when there is large route scale, BRS/UT gives better performance only when there are multiple BGP paths for a given route/prefix. Due to the overheads of multithreading, when you have a single path per route/prefix the system might perform more slowly than the non-multithreaded case.

RIB-FIB Ratio is defined as follows:

`(Total number of BGP routes) / (Total number of unique BGP routes).`

The higher the RIB-FIB ratio, the higher the amount of work done in the shard/update-IO threads in a parallel fashion, and hence the higher the performance gain from multithreading. The RIB-FIB ratio is higher than one when a set of routes are learned from multiple BGP peers.

In this book's lab testing we found that a RIB-FIB ratio of 4:1 or higher shows good performance gains when all other considerations are being met. A low RIB-FIB ratio of 1:1 almost never shows any performance improvements, and in some cases might end up showing a degradation.

NUMA

Non-Uniform Memory Access (NUMA) comes into the picture when systems that have multiple CPU chips, each with their own local memory and I/O channels, combine to form a single pool of multiprocessor CPUs that the operating system manages. Communication and data transfer between the CPU chips are done over high-speed interconnects connecting the chips. Any operation that involves communication between chips takes a longer time than if that operation was done locally within a single chip. For example, sending a message between two threads of execution is much faster if they are executing on the same CPU chip than when they execute on different CPU chips. Thus, how a NUMA system is configured and used for an application has a profound impact on that application's performance.

NUMA considerations for BRS/UT arise only when the RPD is being deployed on third-party server platforms. In such a scenario, ensure that the following factors are considered:

- All CPUs allocated for use by Junos/RPD threads are all from the same NUMA node.
  - The `lscpu` command in Linux can be used to get the mapping of CPUs to NUMA nodes. There are tools available in the VM/container environments within which RPD runs that list and configure the CPUs used for a VM/container. They can be used in conjunction with the `lscpu` command to ensure these NUMA node considerations are met.
- Make sure that the Network Interface Cards (NICs) through which the Junos/RPD control plane traffic is received are also on the same NUMA node as the allocated CPUs.
  - The `cat /sys/class/net/<device-name>/device/numa_node` command on a Linux system lists the NUMA node that a device is attached to. Use this command to get the NUMA node for the NIC being used and verify that the CPUs are also allocated from the same NUMA node.

BGP Grouping

Junos BGP arranges one or more BGP peers into BGP groups. Peers in a BGP group share attributes, configuration parameters, and more. A significant amount of BGP egress pipeline processing happens at the BGP group level. Because of this, BGP grouping becomes important from a performance standpoint.

In order to keep the number of BGP groups to a minimum, combine peers that need similar configuration and treatment into the same group if possible. In general, for a given number of peers, a system with a smaller number of BGP groups scales and performs well as compared to a system that uses larger number of BGP groups. On the flip side, a few groups with very large number of peers in a system where the average size is much smaller can cause certain update-IO threads to handle disproportionately higher loads. In such cases it might be advisable to split these large groups into smaller ones to help spread the load.

With BRS/UT, each BGP group is assigned to a particular update-IO thread for egress pipeline processing. When there are more groups than threads, this assignment uses a *round-robin* scheme to distribute groups to threads. The lower the number of BGP groups that an update-IO thread has to handle, the better the share of processing that each of those groups get.

BGP Families

BRS is supported for a certain set of BGP address families, so check if the address families in your planned deployment are supported. The set of supported address families will grow in future Junos releases, so consult the Junos documentation for your software release to see what is supported and refer to Table 2.1.

Table 2.1 BGP Address Families Supported in Junos 20.1

Platforms	BGP Families Supported
All Platforms except cRPD	INET4 Unicast, INET6 Unicast
cRPD	INET4 Unicast, INET6 Unicast, INET4-VPN Unicast, INET6-VPN Unicast, Route Target

Number of Shard/Update-IO Threads

By default, the number of shard and update-IO threads is set to the number of CPUs available in the underlying system. The number of threads might have to be changed via configuration for several possible reasons:

- Too many threads can impact the availability of CPU resources for other critical processing in the system when there is contention for CPU resources. If it's set to too low a value it might lead to reduced performance when CPU cycles are available, but you do not have enough threads to utilize it. These concerns have to be balanced to arrive at an appropriate value for the number of threads.
    - When there are more threads contending for CPU resources than there are available CPUs, then sharing available CPU resources will be arbitrated by the OS scheduler based on the scheduling scheme that is configured. The CPU time that any thread receives in such a scenario will depend on a lot of other factors.
- The number of threads in use impacts steady state memory usage. This is covered later, but in general you should choose the number of threads to balance against any memory usage thresholds that might be in place.
- In a system handling higher BGP route scale, more shard threads results in better performance during convergence events that involve a high volume of route events. On the other hand, if the route scale is limited and not able to drive shard threads in parallel for longer time durations, reducing the number

of shard threads might be beneficial. Use expected route scales to choose the number of shard threads.

- If there are more update-IO threads than there are BGP groups, then the threads that are in excess will not perform any useful work. They might add overheads that reduce overall performance and scale. You can reduce the number of update-IO threads in such a scenario, but on the other hand, if there are a lot of BGP groups oversubscribing the update-IO threads, then increasing the number of update-IO threads might improve performance.

You can see that a lot of these considerations are interdependent. Be sure to choose the right values for the number of threads based on your priorities for the various factors that are involved.

We recommend that you test different combinations in your lab to arrive at suitable values for the number of shard and update-IO threads in your deployment. (The Junos configuration to be used for setting the number of threads is covered in Chapter 4.)

Other Considerations

Memory Usage

When you enable BRS/UT the steady-state memory used by RPD will increase. This is primarily due to the additional state needed in the shard and update-IO threads for concurrently processing BGP pipeline. The amount of increase depends on a lot of factors. Be sure that the memory usage is within permissible limits for your setup.

Reducing the number of shard/update-IO threads can help reduce memory usage. See Chapter 4 for more details about the Junos configurations to be used. This option has the potential to reduce the performance gains, so a balance needs to be struck.

FIB Download

BRS/UT improves BGP learning and advertisement performance by parallelizing the execution of the BGP processing pipeline. However, the way in which active FIB routes are downloaded to the forwarding plane has not changed with BRS/UT. Due to the way the modules involved interact, it is possible to see a reduction in FIB download rate in some platforms under some conditions. If the FIB download rate is an important consideration for you, ensure that adequate lab testing is done to check for, and measure, any degradation before deploying BRS/UT.

Caveated Features

Some Junos features may not be compatible with BRS/UT and are therefore disallowed when BRS/UT is enabled. The list of such caveated features will change over time as more and more features are made capable of operating with BRS/UT. Be sure to consult the Junos documentation for your release for the list of caveat features before enabling BRS/UT. If you have features that are not needed, then you can delete/deactivate them before enabling BRS/UT.

Checklist for Third-Party Servers

Here's a quick summary of server-specific items that need to be checked before deploying BRS/UT on third-party servers:

- Ensure that the system is a multiprocessor (multicore) system with 64-bit support and enough CPUs available for the shard and update-IO threads. Ensure that Hyper Threading is turned off.
- Configure the system CPUs to operate in performance mode.
- Assign all CPUs for use by Junos/RPD from the same NUMA node in a NUMA system.
- Ensure that I/O performance capabilities like SRIOV are enabled. Ensure that NIC and its driver are tuned for the best performance possible.
- Ensure that NICs through which BGP protocol traffic is received are on the same NUMA node from which CPUs are assigned for Junos/RPD.

Chapter 3

Target Use Cases

In this chapter you will learn about the deployment scenarios that have a very high potential to benefit from BRS/UT. For each scenario you will be introduced to the use case, how a typical deployment for that use case is set up, and what the deployment considerations are before you apply BRS/UT. You will also get an idea of the performance improvements possible from the book's lab testing results for that use case.

Internet Peering

Peering routers, present at the edge of a network, connect with routers from other networks (organizations) using eBGP. To enable easy connectivity to other networks they are physically deployed in Internet Exchanges that host peering routers belonging to different networks and organizations. Peering routers learn addresses reachable via other networks, while addresses owned by, or reachable through, the router's own network are advertised. Some of the salient features of a peering router are:

- Large number of eBGP peers.
- High scale of routes involved (several million RIB entries usually).
- Heavy application of BGP policies for the import/export of routes.

BRS/UT Deployment Considerations

Peering routers have large BGP RIBs with high RIB:FIB ratios. Due to their key role, they also need fast BGP convergence performance. Typical Juniper products

deployed in peering roles (PTX routers in most cases) have multicore Routing Engines (RE). All these factors combine to make peering routers a compelling use case for deploying BRS/UT.

Test Results

Tests were done to measure performance benefits of BRS/UT for a typical peering deployment whose salient features are captured in Table 3.1.

Table 3.1 Peering Router Test Scenario

Platform Used	PTX10003-80C
Software	20.1R2-S3.3-EVO
Route Scale	IPv4: 11M routes total; 1M unique routes RIB:FIB Ratio: 11:1 (IPv4)
BGP Scale	21 BGP Groups 202 BGP Peers: 20 x 10 eBGP IPv4 + 2 x 1 iBGP to RR
RSVP MPLS LSPs	200 Ingress; 200 Egress
Shard/Update Threads Used	12 shard, 12 Update-IO

Configuration snippets highlighting important portions of the Junos configuration that was used are given below:

```
system {

<...>

    processes {
        routing {
            bgp {
                rib-sharding {
                    number-of-shards 12;
                }
                update-threading {
                    number-of-threads 12;
                }
            }
        }
    }
}

<...>

policy-options {
    prefix-list ipv6-plist-1 {
        2001:4860::/32;
        2401:fa00::/32;
        2404:6800::/32;
        2607:f8b0::/32;
        2800:3f0::/32;
```

```
        2a00:1450::/32;
        2c0f:fb50::/32;
    }

<...>

    policy-statement AS12001-non-continental-check {
        term continental-check {
            from community NON-CONTINENTAL-RR;
            then accept;
        }
        term reject-all-other-routes {
            then reject;
        }
    }
    policy-statement AS12001-non-regional-check {
        term regional-check {
            from community NON-LOCAL-RR;
            then accept;
        }
        term reject-all-other-routes {
            then reject;
        }
    }

<...>

}

<...>

protocols {

<...>

    bgp {
        precision-timers;
        path-selection {
            always-compare-med;
            external-router-id;
        }
        advertise-inactive;
        log-updown;

<...>

        group g1 {
            apply-groups BGP-LOCALADDR;
            import [ VERIFIED-RTS ACCEPT-SELECT ];
            export [ DROUTES RROUTES ];
            peer-as 12001;
            neighbor 192.168.0.17 {
            }
            neighbor 2001:4860::192:168:0:17 {
            }
        }

<...>
```

```
        group gr1 {
            local-address 192.168.0.6;
            hold-time 300;
            import excl-filter;
            family inet {
                unicast;
            }
            family inet6 {
                unicast;
            }
            cluster 192.168.0.6;
            peer-as 12001;
        }

<...>

        group ginternal-v4 {
            type internal;
            local-address 192.168.0.6;
            family inet {
                any;
            }
            export PROTRT;
            multipath;
            neighbor 67.0.0.2;
            neighbor 67.0.1.2;
            neighbor 67.0.2.2;
            neighbor 67.0.3.2;
            neighbor 67.0.4.2;
            neighbor 67.0.5.2;
            neighbor 67.0.6.2;
            neighbor 67.0.7.2;
            neighbor 67.0.8.2;
            neighbor 67.0.9.2;
        }
    }

<...>

    mpls {
        statistics {
            interval 300;
            auto-bandwidth;
            no-transit-statistics;
            no-transit-statistics-polling;
            statistics-query-batch-size 6;
        }

<...>

        label-switched-path CSIM-LSP-R6-R2-1 {
            apply-groups CSIM-BB-BB-MPLS;
            to 192.168.0.2;
            primary CSIM-PATH-R2;
        }
        label-switched-path CSIM-LSP-R6-R2-2 {
            apply-groups CSIM-BB-BB-MPLS;
```

```
            to 192.168.0.2;
            primary CSIM-PATH-R2;
        }

<...>

    }

<...>
```

With the test configuration shown above, the Device Under Test (DUT) was allowed to come up and learn all the routes. Once steady state was reached, a `clear bgp neighbor all` command was issued from CLI. This causes all BGP learned route state to be cleaned up and relearned, thereby helping measure route learning performance in a focused way. For accurately measuring performance, traffic was sent through the DUT for a subset of routes involved, with the traffic outage time giving a measure of how fast the relearning happened. With BRS/UT the DUT converged *four times faster* than when BRS/UT was not configured. In other words, the traffic outage time was less than one fourth the outage time seen when BRS/UT was not deployed.

Datacenter Edge

Datacenter (DC) Edge routers act as the link between a DC and the external world. DC Edge routers connect to:

- The DC Fabric of the DC for which external connectivity is provided.
- The core network of the cloud provider for connecting to other DCs and other resources of the provider.
- Peering router for connecting to other service providers and end users. In some scenarios DC Edge routers also assume the peering functionality.

Due to central nature of the DC Edge role these routers peer with a significant number of BGP peers and deal with a large amount of BGP RIB scale.

BRS/UT Deployment Considerations

As mentioned, DC Edge routers typically have a high RIB scale with a reasonably high RIB:FIB ratio. Due to the crucial role they play, fast BGP convergence is critical for them. Juniper's MX platforms, which are often deployed in DC Edge roles, have multicore support. Thus, DC Edge routers are a good use case for deploying BRS/UT, as it meets all the important deployment considerations.

Test Results

The key parameters used for testing the DC Edge router use case are given in Table 3.2.

Table 3.2 DC Edge Router Testing Scenario

Platform Used	MX480, RE-S-2X00x6, MPC7E 3D
Software	20.1R1.11
Route Scale	IPv4: 7M routes total; 1M unique routes IPv6: 600K routes total: 100K routes unique RIB:FIB Ratio: 7:1 (IPv4), 6:1 (IPv6)
BGP Scale	8 BGP Groups 1104 BGP Peers: 2 x 450 eBGP IPv4 + 2 x 100 eBGP IPv6 + 4 x 1 iBGP to RR
RSVP MPLS LSPs	100K Transit; 15K Ingress; 15K Egress
Shard/Update Threads Used	2 shard, 2 Update-IO

You can see some important snippets from the configuration used for the tests below:

```
system {

<...>

    processes {
        routing {
            bgp {
                rib-sharding {
                    number-of-shards 2;
                }
                update-threading {
                    number-of-threads 2;
                }
            }
        }
    }
}

<...>

protocols {
    rsvp {
        interface lo0.0;
        interface ae1.0;
        interface ae11.0;
        interface ae2.0;
        interface ae22.0;
        interface xe-3/0/0:2.0;
        interface xe-0/1/2:2.0;
```

```
    }
    mpls {
        traffic-engineering {
            mpls-forwarding;
        }
        explicit-null;
        ipv6-tunneling;
        label-switched-path R52R0-1 {
            to 10.255.33.188;
        }
        label-switched-path R52R0-2 {
            to 10.255.33.188;
        }
        label-switched-path R52R0-3 {
            to 10.255.33.188;
        }

<...>

        interface ae1.0;
        interface ae11.0;
        interface ae2.0;
        interface ae22.0;
        interface lo0.0;
        interface xe-0/1/2:2.0;
    }
    bgp {
        precision-timers;
        path-selection always-compare-med;
        traceoptions {
            file bgp.log size 10m files 10 world-readable;
        }
        advertise-inactive;
        log-updown;
        local-as 8075;
        group 8075-STD-FREEPUBLIC-PEERGROUP {
            type external;
            description "Public Peers";
            import COLOR-EBGP-ROUTES;
            family inet {
                unicast {
                    prefix-limit {
                        maximum 6000000;
                        teardown 90 idle-timeout 45;
                    }
                }
            }
            export [ default-originate-ebgp DENY-ALL ];
            multipath multiple-as;
            neighbor 20.1.1.2 {
                #authentication-key "$9$I9RRyevMLNVsfTyeWxVb.mf5Q3/CpIRc7-DH.PF3Atu0BR"; ## SECRET-
DATA
                peer-as 10000;
            }
            neighbor 20.1.2.2 {
                #authentication-key "$9$hDIceM8L7VsgQFeMX-sYf5Qz3/ApOhcrdbH.fT6/u0B1Rc"; ## SECRET-
DATA
                peer-as 10001;
            }
```

```
<...>

    }
    isis {
        apply-groups default_isis_setup;
        traffic-engineering {
            family inet6 {
                shortcuts;
            }
            family inet {
                shortcuts;
            }
        }
        interface xe-0/1/2:2.0;
        interface xe-3/0/0:2.0 {
            point-to-point;
            level 1 disable;
            level 2 metric 1000;
        }

<...>

        interface lo0.0;
    }
    ldp {
        apply-groups default_ldp_setup;
        track-igp-metric;
        egress-policy export-lo0-in-ldp;
        interface lo0.0;
        session 10.255.33.188;
    }

<...>

policy-options {
    prefix-list permit_static_bgp_anchor {
        10.43.240.0/20;
        199.242.32.0/23;
    }
    prefix-list deny_static_bgp;
    prefix-list permit_static_bgp {
        207.46.32.132/32;
    }
    prefix-list permit_connected_bgp_exceptions {
        20.0.0.0/8;
        207.46.33.32/28;
    }

<...>

    policy-statement COLOR-EBGP-ROUTES {
        from protocol bgp;
        then {
            color 20;
            next policy;
        }
    }

<...>
```

```
    policy-statement IPV6-CUST-CIS-PREFIX-LIST {
        term 100 {
            from {
                route-filter 2a01:111:e100::/46 orlonger;
                route-filter 2a01:111:f100::/46 orlonger;
                route-filter 2603:1000::/29 orlonger;
                route-filter 2603:1010::/29 orlonger;
                route-filter 2603:1020::/29 orlonger;
                route-filter 2603:1030::/29 orlonger;
                route-filter 2603:1040::/29 orlonger;
                route-filter 2603:1050::/29 orlonger;
                route-filter 2603:1080::/29 orlonger;
                route-filter 2603:1090::/29 orlonger;
                route-filter 2603:10a0::/29 orlonger;
                route-filter 2603:10b0::/29 orlonger;
                route-filter 2603:10c0::/29 orlonger;
                route-filter 2603:10d0::/29 orlonger;
            }
            then accept;
        }
        then accept;
    }

<...>
```

In the DC Edge router the `clear bgp neighbor all` event described in detail in the *Internet Peering* use case was triggered. With BRS/UT, the DUT was able to relearn and converge about *two-and-a-half times faster* than when the BRS/UT is not configured.

Route Reflector

Route Reflectors (RR) help simplify iBGP deployments by reducing the number of BGP peering sessions required. By acting as an intermediary between BGP speakers, they facilitate BGP learning and advertisement in a scalable and efficient manner in networks having a large number of iBGP nodes. RRs are a perfect use case for deploying BRS/UT, all other considerations being met. RRs typically peer with a large number of BGP clients and handle very high route scale, typically in the order of several million to tens of millions of routes. The RIB:FIB ratio is quite high in most RR deployments. Most RRs are deployed in an out-of-band manner where they do not participate in packet forwarding, which makes FIB download performance less relevant for them.

Juniper's RR offerings are available for deployment in a variety of form factors:

- vRR: virtualized implementation designed to run on Virtual Machines (VMs) on third-party server platforms.
- cRR: cRPD deployed in RR role on standard container environments on third-party server platforms.

- JRR: virtualized RR implementation running on Juniper-provided appliance.
- Juniper Router Platforms: RR functionality can be enabled on most of Juniper's routers via configuration.

Almost all deployments use a multicore/multi-processing capable system with a good amount of memory needed to support the route scales that are involved.

BRS/UT Deployment Considerations

All the important considerations like RIB scale, RIB:FIB ratio, and BGP peer scale should all be satisfied in typical RR deployments. While platform considerations like multi-processing support should be taken care of for Juniper router platforms, for vRR and cRPD deployments on third-party servers, ensure that all the recommendations listed in Chapter 2 are met.

Test Results

Testing for RR use case was done on the following four RR platforms:

- MX204 as an RR
- JRR200
- vRR on a standard server
- cRR on a standard server

The tests involved learning and advertising real world (internet) route feeds. Route scales were chosen per platform capabilities. The test setup was designed to ensure that nodes other than the device under test were not a bottleneck. Convergence performance was measured for real world convergence scenarios with and without BRS/UT, and the results were compared. The results are provided for each of the four RR platforms.

Snippets from the Junos configuration used in the RR system are given below for your reference. All the RR platforms used a similar configuration with the only difference being the scale of BGP peers, groups, and routes. Of course, the number of shard and update-IO threads for optimal performance could also be different depending on the platform.

```
<...>

system {

<...>

    processes {
        routing {
            bgp {
                rib-sharding {
```

```
                    number-of-shards 5;
                }
                update-threading {
                    number-of-threads 5;
                }
            }
        }
    }
}

<...>

routing-options {
    autonomous-system 1;
}
protocols {
    bgp {
        family inet {
            unicast {
                no-install;
            }
        }
        group intnl {
            type internal;
            neighbor 8.1.1.20 {
                local-address 8.1.1.10;
            }
            neighbor 8.1.1.21 {
                local-address 8.1.1.11;
            }

<...>

            neighbor 8.1.1.29 {
                local-address 8.1.1.19;
            }
        }
        group g1 {
            type internal;
            cluster 1.1.1.2;
            neighbor 1.30.1.1 {
                local-address 1.20.1.1;
            }
            neighbor 1.30.1.2 {
                local-address 1.20.1.2;
            }

<...>

            neighbor 1.30.1.49 {
                local-address 1.20.1.49;
            }
            neighbor 1.30.1.50 {
                local-address 1.20.1.50;
            }
        }

<...>
```

```
        group g20 {
            type internal;
            cluster 1.1.1.2;
            neighbor 1.30.20.1 {
                local-address 1.20.20.1;
            }
            neighbor 1.30.20.2 {
                local-address 1.20.20.2;
            }

<...>

            neighbor 1.30.20.49 {
                local-address 1.20.20.49;
            }
            neighbor 1.30.20.50 {
                local-address 1.20.20.50;
            }
        }
        hold-time 300;
        mtu-discovery;
    }
    ospf {
        area 0.0.0.0 {
            interface ge-1/0/1.0;
            interface ge-1/0/1.5;
            interface lo.0 {
                passive;
            }
        }
    }
}
```

MX204

The MX204 is a compact routing platform built to comply with service provider (SP) network requirements with support for a wide variety of external connecting ports. This makes MX204 a platform of choice for deployment as an RR (see Table 3.3).

Table 3.3 MX204 as RR

Platform Used	MX204, RE-S-1600x8
Software	20.1R1.11
Route Scale	6.4 M routes total; 800 K unique routes 8:1 RIB:FIB Ratio
BGP Scale	21 BGP Groups; 168 BGP Peers (21 x 8) Learning from: 8 peers Advertising to: 160 peers
Shard/Update Threads Used	4 shard, 4 Update-IO

In the test configuration, time was measured to learn all the routes from scratch and advertise them fully, a typical convergence scenario that happens when an RR comes online (after a reboot). Convergence with BRS/UT took just *two-thirds of the time* it takes when BRS/UT is not configured.

Another convergence scenario occurred when BGP peers flap due to network events during which peers go down and come back up. This results in relearning and the readvertisement of routes. This scenario was tested with an increased scale of 200 downstream peers (peers to which RR advertises to) by flapping 150 of those peers randomly. Convergence with BRS/UT took *less than half the time* (was more than twice as fast) with five shards and update-IO threads.

JRR200

JRR200 is a Juniper appliance devised exclusively to be used as a route reflector. Shown in Table 3.4, it has a multicore CPU and the large RAM memory needed for the high route scales that an RR typically handles, along with Ethernet interfaces for external connectivity. Juniper's vRR software runs in a VM inside this system.

Table 3.4 JRR200

Platform Used	JRR200
Software	20.1R1.11
Route Scale	8 M routes total; 800 K unique routes 10:1 RIB:FIB Ratio
BGP Scale	21 BGP Groups; 250 BGP Peers (1 x 10 + 20 x 12) Learning from: 10 peers Advertising to: 240 peers
Shard/Update Threads Used	10 shard, 10 Update-IO

In Junos 20.1, only six cores are allocated for the VM in which the vRR software runs. This test was done with a VM configuration change to allocate ten cores to this VM. This expanded allocation will be made available to our customers in an upcoming Junos release.

For the complete learning and advertisement convergence scenario, BRS/UT took about *three-fourths of the time* when compared to no BRS/UT baseline.

With BRS/UT, when 75% of downstream peers were flapped, convergence was achieved in *60% of the time* taken for baseline scenario.

The results would have been better if not for the virtualized I/O performance within JRR, which uses a slower software virtualization implementation. SRIOV support is being planned for in an upcoming Junos release, and you can expect to see even better results with BRS/UT.

vRR

Virtual Route Reflector (vRR) is a Junos OS software package to be deployed on VMs for use as a route reflector and is supported for most of the popular VM flavors used in the industry. From a BRS/UT perspective, the capabilities of the server in which these VMs run matters the most. This was covered in detail in Chapter 2. The server used for this testing along with other details on the test scenario are captured in Table 3.5.

Table 3.5 vRR

Platform Used	24-core Intel(R) Xeon(R) CPU E7-8890 v4 @ 2.20GHz 256GB memory
Software	20.1R1.11
Route Scale	8 M routes total; 800 K unique routes 10:1 RIB:FIB Ratio
BGP Scale	21 BGP Groups; 1010 BGP Peers (1 x 10 + 20 x 50) Learning from: 10 peers Advertising to: 1000 peers
Shard/Update Threads Used	24 shard, 24 Update-IO

You can see in this first convergence scenario of learning and advertising all routes from scratch BRS/UT was *more than four times faster* than the baseline (took *less than one fourth the time taken* by the baseline).

When 75% of the 1000 downstream peers were flapped the system was able to converge in about the *half the time taken* by the baseline with no BRS/UT. This was observed with six shards and six update-IO threads configured.

cRR

Containerized Route Reflector (cRR) is a cRPD that is deployed as a BGP Route Reflector. cRPD is available as a Junos OS software package that can be deployed in standard container environments like Docker. From a BRS/UT perspective, the capabilities of the server in which these containers run matter the most. This was covered in detail in Chapter 2. The server used for this testing, along with other details on the test scenario, are captured in Table 3.6.

Table 3.6 cRR

Platform Used	24-core Intel(R) Xeon(R) CPU E7-8890 v4 @ 2.20GHz 256GB memory
Software	20.1R2-S1.1
Route Scale	8 M routes total; 800 K unique routes 10:1 RIB:FIB Ratio
BGP Scale	21 BGP Groups; 1010 BGP Peers (1 x 10 + 20 x 50) Learning from: 10 peers Advertising to: 1000 peers
Shard/Update Threads Used	12 shard, 12 update-IO

In the first convergence scenario of learning and advertising routes from scratch, BRS/UT was *almost nine times faster* than the baseline (meaning it took only about one tenth of the time taken by baseline)!

When 75% of the 1000 downstream peers were flapped at random, the system was able to converge almost *four times faster* than the baseline. This result was observed with 10 shards and 10 update-IO threads.

Chapter 4

Configuration and Operational Aspects

This chapter focuses on how to enable/disable BRS/UT in a system via a Junos configuration and how to configure the number of shard/update-IO threads to be used. You will also learn some operational commands that you can use to get useful information about these features and monitor how they are functioning.

Configuration

BRS/UT is not enabled by default. Use the following Junos configuration knob to enable BRS/UT:

```
system {
    processes {
        routing {
            bgp {
                rib-sharding {
                    number-of-shards 4;
                }
                update-threading {
                    number-of-threads 8;
                }
            }
        }
    }
}
```

Use the `number-of-shards` and `number-of-threads` options to set the number of shard and update threads, respectively, if needed. Be sure to consider all the factors outlined in Chapter 2 for choosing the appropriate number of threads.

NOTE Consult the Junos documentation for your specific software release for more details on the above configuration knobs.

Some key things to note:

- BRS/UT can only be enabled on the default master logical system. It requires that the master logical system be operating in 64-bit mode.
- While Update Threading can be enabled independently, enabling RIB Sharding requires that UT also be enabled.
- Configuring or unconfiguring BRS/UT results in a restart of the RPD daemon. Be sure to plan ahead to mitigate the impact of this.

Configuring Update-Threading Alone

UT can be enabled without BRS. This is done by omitting the `rib-sharding` knob in the configuration stanza. In this mode, the BGP pipeline operations performed by shard threads will be done by the main thread now. Update-IO threads will handle their part of the pipeline – update PDU generation and transmission. This mode of operation will only speed up the egress pipeline activities done in update-IO thread context.

In this mode the main thread can become a bottleneck in convergence scenarios. This can in turn result in the update-IO threads not being fully utilized, thereby limiting the performance gains.

Operational Commands

Use the commands outlined here to monitor how BRS/UT is operating.

BRS/UT status - `show bgp summary`

Use this command to see if BRS/UT is enabled and to get the number of threads in use:

```
root@r1_re> show bgp summary

Threading mode: BGP sharding
Thread counts: Update-io: 8 Shards: 4
Default eBGP mode: advertise - accept, receive - accept
Groups: 1 Peers: 1 Down peers: 1
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                       0          0          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.1.1.2                  2          0          0       0      12           5 Idle
```

Shard/Main Route Counts – `show route summary`

Use this command to get the BGP route counts in shard threads and in the main thread. For each route table that has been sharded, this command will show the aggregate route count and also a breakup of the route counts from each thread:

```
root@R1_re> show route summary
Autonomous system number: 1
Router ID: 1.1.1.1

Highwater Mark (All time / Time averaged watermark)
    RIB unique destination routes: 803654 at 2020-07-31 14:05:06 / 0
    RIB routes                   : 803655 at 2020-07-31 14:05:06 / 0
    FIB routes                   : 803614 at 2020-07-31 14:05:06 / 0
    VRF type routing instances   : 0 at 2020-07-31 13:46:46

inet.0: 803621 destinations, 8035769 routes (803621 active, 0 holddown, 0 hidden)
              Direct:     14 routes,     14 active
               Local:      2 routes,      2 active
                 BGP: 8035720 routes, 803572 active
              Static:     33 routes,     33 active

junos-main::inet.0: 803621 destinations, 803621 routes (803621 active, 0 holddown, 0 hidden)
              Direct:     14 routes,     14 active
               Local:      2 routes,      2 active
                 BGP: 803572 routes, 803572 active
              Static:     33 routes,     33 active

junos-bgpshard0::inet.0: 200603 destinations, 2005841 routes (200603 active, 0 holddown, 0 hidden)
                 BGP: 2005820 routes, 200582 active

junos-bgpshard1::inet.0: 199915 destinations, 1998907 routes (199915 active, 0 holddown, 0 hidden)
                 BGP: 1998880 routes, 199888 active

junos-bgpshard2::inet.0: 201453 destinations, 2014314 routes (201453 active, 0 holddown, 0 hidden)
                 BGP: 2014290 routes, 201429 active

junos-bgpshard3::inet.0: 201698 destinations, 2016755 routes (201698 active, 0 holddown, 0 hidden)
                 BGP: 2016730 routes, 201673 active

__raass__inet.inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
               RaaS:     10 routes,     10 active

iso.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
              Direct:      1 routes,      1 active
<…>
```

The aggregate and thread-wise breakup of counts will be displayed only for inet.0 and inet6.0 as these are the only tables containing routes handled by BRS today. This applies for other commands listed here as well.

Shard/Main Route Table View – `show route rib-sharding`

A new `rib-sharding` option was introduced for the `show route` command as part of BRS/UT. Use this option to see the route table entries for a specific thread – any of the shard threads, or the main thread. Here's sample output for a main thread:

```
root@R1_re> show route rib-sharding main

inet.0: 803721 destinations, 803721 routes (803721 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.0.0.0/24         *[BGP/170] 01:53:57, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
1.0.4.0/22         *[BGP/170] 01:55:37, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
1.0.4.0/24         *[BGP/170] 01:55:37, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
1.0.5.0/24         *[BGP/170] 01:55:37, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
1.0.6.0/24         *[BGP/170] 01:55:37, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
<…>
```

And here's a sample output for a shard thread:

```
root@R1_re> show route rib-sharding junos-bgpshard2

inet.0: 201468 destinations, 2014464 routes (201468 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.0.0.0/24         *[BGP/170] 01:56:29, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.21
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.22
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.23
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.24
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.25
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.26
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.27
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.28
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:56:29, localpref 100, from 1.1.1.29
                      AS path: 69 10458 14203 2914 13335 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
```

```
1.0.4.0/24         *[BGP/170] 01:58:09, localpref 100, from 1.1.1.20
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:58:09, localpref 100, from 1.1.1.21
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
                    >  to 10.1.1.2 via ge-0/0/0.0
                     [BGP/170] 01:58:09, localpref 100, from 1.1.1.22
                      AS path: 69 10458 14203 4826 38803 56203 I, validation-state: unverified
<...>
```

When the rib-sharding option is not specified, this command will show the aggregate view of route table.

Other RIB Sharding commands are capable of providing aggregate and thread-specific information that you may find useful:

```
show bgp summary

show bgp neighbor

show bgp group
```

Knowing the Update-IO Thread for a BGP Group – `show bgp group`

Use the `show bgp group` command to see the specific update-IO thread that is servicing a BGP group:

```
root@R1_re> show bgp group

Group Type: Internal    AS: 1                     Local AS: 1
  Name: intnl           Index: 0                  Flags: <Export Eval>
  Update thread: bgp-updio-0
  Options: <GracefulShutdownRcv>
  Holdtime: 0
  Graceful Shutdown Receiver local-preference: 0
  Total peers: 10       Established: 10
  1.1.1.20+179
  1.1.1.21+57693
  1.1.1.22+61325
  1.1.1.23+179
  1.1.1.24+179
  1.1.1.25+65054
  1.1.1.26+61345
  1.1.1.27+58376
  1.1.1.28+53799
  1.1.1.29+179
  Number of peers closing: 0
  inet.0: 803624/8036240/8036240/0

Groups: 1  Peers: 10   External: 0    Internal: 10   Down peers: 0   Flaps: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
inet.0
                 8036240     803624          0          0          0          0
```

Be sure to consult the Junos documentation for your specific Junos release to get the complete list of configuration and operational commands for BRS/UT.

Chapter 5

Conclusion

Let's summarize the key take-aways and offer a quick overview of the BRS/UT features that will be delivered in upcoming Junos releases.

Deploying BRS/UT

Be sure to look carefully at all the factors for your specific deployment scenario. Ensure that all the considerations are met and that there is a case for deploying and benefitting from BRS/UT. If you choose to deploy, you might still have to work out the right configuration parameters as the defaults may not be the right ones for your scenario. Be sure to test out various combinations in a lab environment to arrive at the right configuration for your scenario. Prior lab testing before deployment in a production network will help you avoid any surprises and verify that your system is configured to maximize benefits from all of BRS/UT's features.

What's Ahead

To ensure that more and more customers deploy BRS/UT and enjoy the superior BGP convergence performance it brings, support for more BGP address families with BRS/UT will be added in forthcoming Junos releases. Support will also be added for caveat features to widen the deployment opportunities for customers. Future Junos releases will also aim to reduce overheads like memory and increase deployment opportunities.

Here are some scripts and tools that will aid your decisions on some of sharding's deployment considerations. These scripts/tools are available on GitHub. Be sure to check in the future for new additions and updates, here: https://github.com/juniper/bgp-sharding.