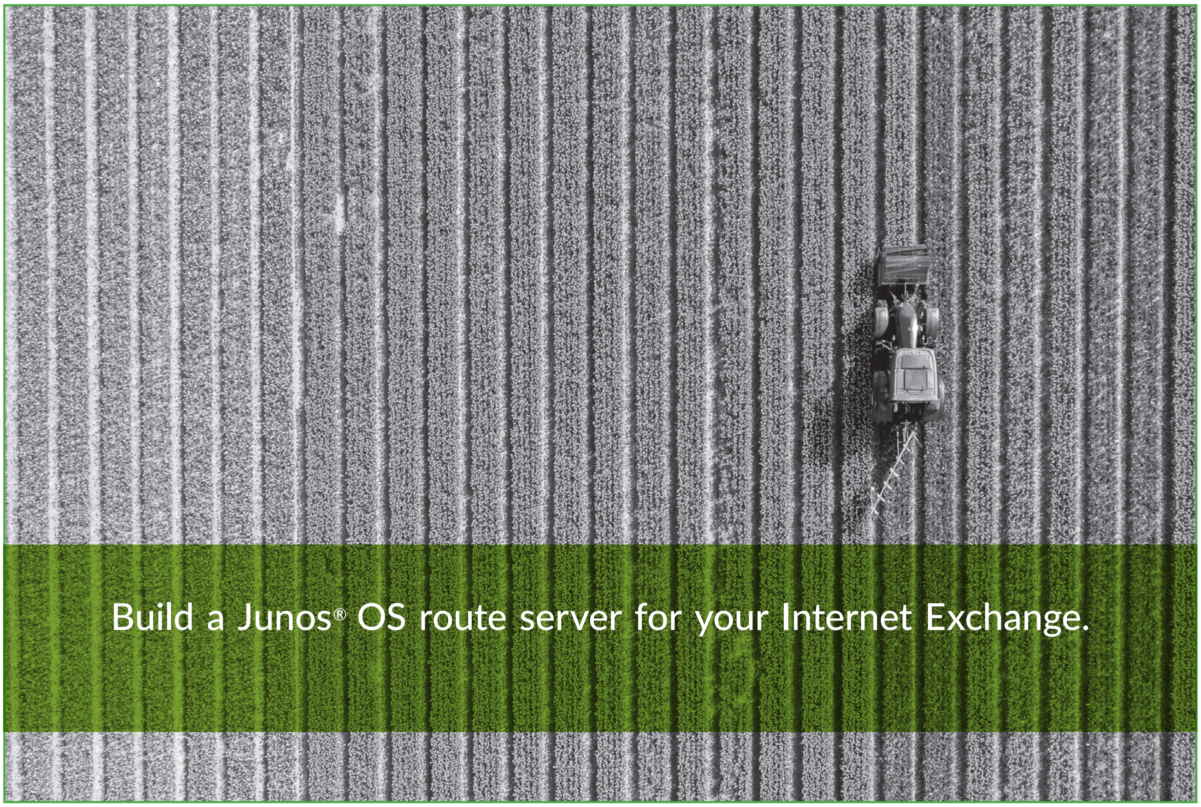
Engineering Simplicity

Day One Million

DAY ONE: DEPLOYING JUNOS® ROUTE SERVERS



By Colby Barth and Melchior Aelmans

DAY ONE: DEPLOYING JUNOS® ROUTE SERVERS

This book targets network engineers, designers, and architects who are involved in Internet Exchange Point (IXP) operations and intend to deploy Junos® OS-based route servers. The Junos OS has included EBGP Route Server requirements since 17.4R1, and is considered a feature rich, high-performing route server implementation. Barth and Aelmans provide Junos implementation tips and tricks, as well as general Internet Exchange configuration considerations such as the usage of policies, communities, and routing security methods like IRR/RPKI. Also covered are RS scaling, filtering, and validation issues.

"Whether you've been running an IXP for quite some time or planning to start one, this Day One book helps you deploy your route servers – a vital service that exists in every IXP network and that handles prefix propagation between members. The authors include all their field knowledge and best practices to set up new versions of route servers for not only the beginner but also for more experienced engineers to smoothly transition from simple instances to secure deployments that will be MANRS compliant."

Stavros Konstantaras, NOC Engineer, AMS-IX

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

- Deploying route servers in an Internet Exchange Point (IXP).
- What IXP relevant features are offered by the Junos OS.
- Deploying and implementing physical/virtual/containerized (redundant) Junos OS route servers.
- Implementing routing policies and methods that reject invalid routing information in an IXP environment.
- Verifying your configuration and supporting Junos OS route servers using troubleshooting commands.

ISBN 978-1941441961
51600
9 781941 441961

Day One: Deploying Junos® Route Servers

by Colby Barth and Melchior Aelmans

Published by Juniper Networks Books
Author: Colby Barth, Melchior Aelmans
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
Illustrator: Karen Joice

ISBN: 978-1-941441-96-1 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-941441-97-8 (ebook)

Version History: v1, Sept. 2019
2 3 4 5 6 7 8 9 10

http://www.juniper.net/dayone

About the Authors

Colby Barth is a Distinguished Engineer at Juniper Networks. Colby has over 20 years of experience designing, building, and operating IP/MPLS networks.

Melchior Aelmans is a Senior Systems Engineer at Juniper Networks, where he has been working with many operators on the design, security, and evolution of their networks. He has over 10 years of experience in various operations, engineering, and sales engineering positions with enterprises, data centers and Service Provider. Before joining Juniper Networks, he worked with eBay, LGI, KPN, etc. Melchior enjoys evangelizing and discussing topics like BGP, peering, routing security, and Internet routing.

Authors Acknowledgments
The authors would like to thank, in random order: Andrew Gallo (George Washington University); Lucas van den Berg (NL-ix); Martin J. Levy (Cloudflare); Stavros Konstantaras (AMS-IX); Anurag Khare, Lenny Giuliano, and Patrick Ames (Juniper Networks); Job Snijders (NTT Communications); and Coloclue and Internet Society (MANRS initiative) for their valuable contributions, content, input, comments, and emotional support.

Feedback? Comments? Error reports? Email them to dayone@juniper.net.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books. *Day One* books cover the Junos OS and Juniper Networks network-administration with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow. 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
- Ebooks are available at the Apple iBooks Store
- Purchase the paper edition at Vervante Corporation (www.vervante.com) for between $15-$40, depending on page length

A Key Junos Route Server Resource

The Juniper TechLibrary has been supporting Junos route servers with its excellent documentation as part of its *BGP Feature Guide*. This book is not a substitute for that body of work, so you should take the time to review the documentation here: https://www.juniper.net/documentation/en_US/junos/topics/concept/bgp-route-server-overview.html.

What You Need to Know Before Reading This Book

Before reading this book, you should have basic knowledge of IXP operations, some knowledge of how to use route reflectors or route servers, and the how the BGP protocol works.

You should be familiar with the basic administrative functions of Junos OS, including the ability to work with operational commands and to read, understand, and change configurations. There are several books in the *Day One* library on learning Junos at http://www.juniper.net/books.

This book assumes that you, the reader, have an intermediate level knowledge of:

- Junos OS and its command line interface (CLI).
- General BGP protocol use in Internet Service Provider networks.
- General troubleshooting techniques for Internet Service Provider networks running the Junos OS.
- The configuration of basic BGP connectivity in the Junos OS, including configuring neighbors and routing policy.
- Basic Junos OS network and system operation.

What You Will Learn by Reading This Book

This book will help you learn:

- Deploying route servers in an Internet Exchange Point (IXP)
- What IXP relevant features are offered by the Junos OS
- Deploying and implementing physical/virtual/containerized (redundant) Junos OS route servers
- Implementing routing policies and methods that reject invalid routing information in an IXP environment
- Verifying your configuration and supporting Junos OS route servers using troubleshooting commands

About This Book

This book targets network engineers, designers, and architects who are involved in Internet Exchange Point (IXP) operations and intend to deploy Junos OS-based route servers.

Beginning with release 17.4R1, Junos OS includes 'EBGP Route Server' required features:

"Starting in Junos OS Release 17.4R1, BGP feature is enhanced to support EBGP route server functionality. A BGP route server is the external BGP (EBGP) equivalent of an internal BGP (IBGP) route reflector that simplifies the number of direct point-to-point EBGP sessions required in a network. EBGP route server propagates unmodified BGP routing information between external BGP peers to facilitate high scale exchange of routes in peering points such as Internet Exchange Points (IXPs). When BGP is configured as a route server, eBGP routes are propagated between peers unmodified, with full attribute transparency (NEXT_HOP, AS_PATH, MULTI_EXIT_DISC, AIGP, and Communities)." - Source: Junos Release Notes

After reading this book you will be able to deploy a scalable Junos OS based route server, configure advanced route server features, define policies, and incorporate routing security into your route server setup.

This book also contains thoughts and considerations on different deployment scenarios. These are not meant to push you in a specific direction but to offer 'food for thought.'

Last but not least, this book describes how to deploy and use Juniper's containerized routing protocol daemon (cRPD).

Chapter 1

Internet Exchange Point Overview

An Internet Exchange Point (IXP) is a Layer 2 network, for example a Layer 2 MPLS-based service, like VPLS or EVPN, that facilitates interconnection between Internet Service Providers (ISPs) using the Border Gateway Protocol (BGP) protocol to exchange routing information.

At its core, an IXP is essentially one or more physical locations containing interconnected switches that move traffic between the different connected networks (generally referred to as *members* in an IXP context). The network is referred to as the *IXP LAN* or *peering LAN*. See Figure 1.1.

Members share the costs of maintaining the physical infrastructure and associated services via various charging schemes, but in almost every case the membership costs are a fixed monthly fee depending on the port speed and number of ports a member uses.

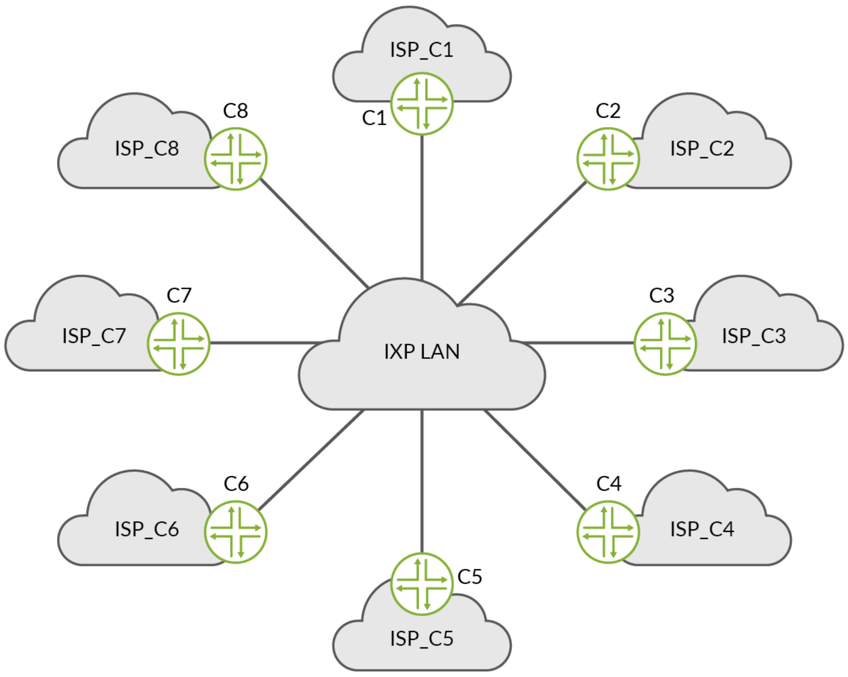


Figure 1.1 *IXP LAN*

Traffic exchange between two members on an IXP is facilitated by the BGP routing configurations between them (peering session). Members choose to announce routes via the peering relationship – either routes to their own addresses, or routes to addresses of other networks that they connect to (for example, customers).

The other party to the peering relationship can then apply route filtering where it chooses to accept those routes and route traffic accordingly, or ignore those routes and use other routes to reach those addresses.

Which routes to advertise, or advertisements to accept or filter from a member, is covered in greater detail in *Day One: Deploying BGP Routing Security* (see: https://www.juniper.net/us/en/training/jnbooks/day-one/deploying-bgp-routing-security/).

The Economics of an Internet Exchange Point

You could say that the goal of an IXP is to reduce the portion of traffic an ISP will deliver via their upstream transit providers, thereby, amongst other things, potentially reducing the average per-bit delivery cost of traffic. Furthermore, the increased number of available paths improves routing efficiency and fault tolerance. Additionally, and sometimes more importantly, goals could be reducing latency, providing shorter network paths, and increasing or providing redundancy.

IXPs exhibit the characteristics of what economists call the *Network Externality Effect* (https://en.wikipedia.org/wiki/Network_effect), or, *the value of the product or service is proportional to the number of users of the product or service*. Internet exchanges are a special case of this effect; the value of an exchange point is not purely the number of participants, but a slightly more complex calculation including the number and uniqueness of the routes and the volume of traffic peered. Since the value of the IXP to an ISP is proportional to the amount of traffic the ISP can exchange in peering relationships at the IXP, the value of the IXP to the peering population follows the network externality graph as shown in Figure 1.2.

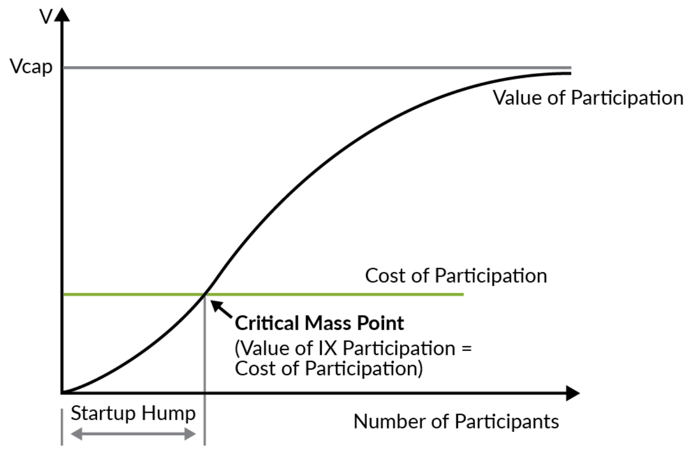


Figure 1.2 Value of an IXP

Figure 1.2 illustrates a plot of the value of the IXP (Vcap) as a function of the number of participants. As more participants connect to the exchange point, more participants can peer with each other. From the point of view of the participants, the value of the exchange point increases with each potential peer.

Peering Relationships

While *bilateral peering* (negotiated and established between exactly two members) sessions were previously the most common means of exchanging routes, the overhead associated with dense interconnection can cause substantial operational scaling problems for members of larger IXPs.

Multilateral interconnection is a method of interconnecting members using a “external brokering” system, commonly referred to as a *route server* and typically managed by the IXP. Each multilateral interconnection participant (referred to as a *route server client*) announces its routes to the route server using External Border Gateway Protocol (EBGP). The route server, in turn, transparently forwards this information to each route server client connected to it.

What is a Route Server?

The barrier of entry for an organization to become a member and begin peering on an IXP is generally quite low. At least a single physical or remote (backhaul) port connected to the IXP peering LAN and an assigned IP address from the IX LAN subnet is needed. It is then possible to configure BGP peering with anyone else on the peering LAN who is willing to peer.

To do so, you have to manually configure those BGP peering sessions with whomever you wish to peer with. This is called *bilateral interconnection*, meaning there is one session between you and the peer, and only you and the specific peer exchange route advertisements.

Imagine a large IXP with many (100+) members. This can very quickly become cumbersome. In particular when filtering per peer on correct route advertisements (for example based on an Internet Routing Registry: https://en.wikipedia.org/wiki/Internet_Routing_Registry), prefixes, AS-path, etc., which you should do. It is a no-brainer to use an easier solution when one is available.

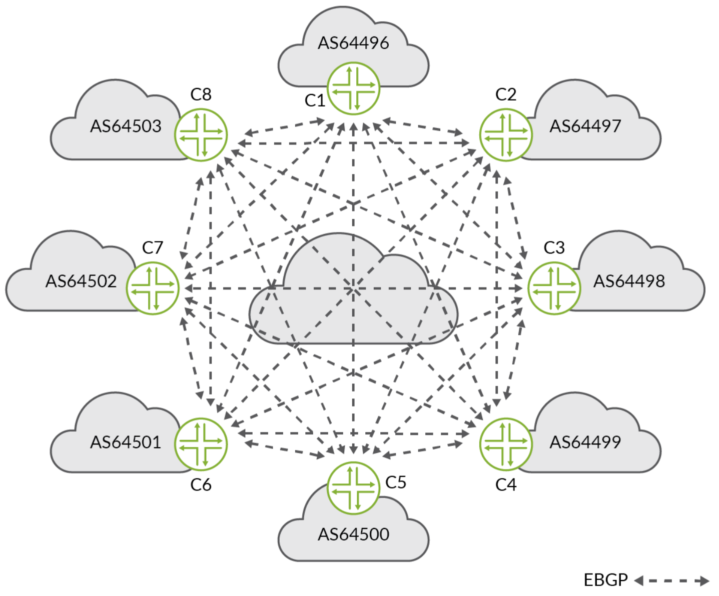


Figure 1.3 *EBGP at an IXP LAN without a Route Server*

To avoid configuring and maintaining 10s or 100s (some of the larger IXPs have over 500 members or members with multiple connections) of individual EBGP sessions with each member, the smarter option where you only need a couple of sessions, is to use the *route servers*. Route servers are typically offered by the IXP as a service to its members.

One BGP session between the ISP's router and the route server is all that is required to announce to, and receive routes from, all other members of the IXP (as shown in Figure 1.4). Obviously, you will only be able to exchange routing information with members who also have a BGP session with the route servers. Thus it provides an alternative to full mesh peering among the members who have a presence at the IXP.

Some ISPs will rely solely on the route server session(s), while others will use it as a backup to their existing EBGP peerings on the IXP fabric. Most IXPs will have redundant route servers and suggest that members peer with both.

Peering with the IXP's route servers could offer the member additional benefits such as *communities* to filter on. For example, their originating country or the originating data center.

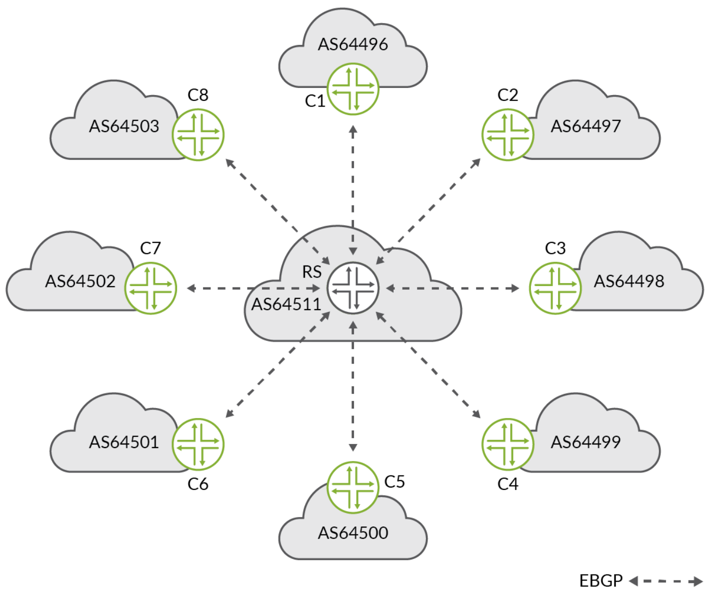


Figure 1.4 EBGP at an IXP LAN with a Route Server

The route server provides:

- EBGP route reflection with customized policy support for each service provider at the IXP.
- Reduced configuration complexity (thus maintaining just a few BGP sessions instead of hundreds).
- Reduced CPU and memory requirements on each member router; you will still receive all the prefixes but won't need all the separate BGP peering sessions.
- Reduced administrative overhead expense incurred by individualized peering agreements.
- Additional filter options (IRRdb, RPKI, predefined BGP communities) without the need to implement those yourself.
- Ability to send and receive traffic via the IXP from day one (no need to wait for all the individual peerings to be arranged).
- A possible backup path; when your BGP session to another member becomes inactive, there is a possibility that you can still reach the members network via routes learned from the route servers.

NOTE The route server itself does not participate in actual traffic forwarding, it only provides routing information (AS-PATHs, routes, communities, next hops, etc.). From that perspective, as no forwarding hardware is needed, a route server can easily be a virtual machine (VM) or container instead of a physical box. Also, peering with a route server does not mean that you have to accept all routes from all other route server participants, or that you have to advertise all your routes to every other member. *How do you configure that?* Read on…

BGP Routing Information Base (RIB)

Throughout this book different types of routing information bases (RIBs) are mentioned. This paragraph prepares you for those. Generally speaking, a RIB holds routing information but there are three RIBs within a BGP speaker that are relevant:

1. *Adj-RIB-In*: Stores routing information learned from inbound BGP UPDATE messages. It contains routes that are available as an input to the BGP decision process.
2. *Loc-RIB*: Contains routing information after applying import policies to the routing information stored in Adj-RIBs-In.
3. *Adj-RIB-Out*: Stores the information that is selected for advertisement to peers. The routing information stored in the Adj-RIB-Out will be used in outbound UPDATE messages and advertised to its peers.

In summary, the Adj-RIB-In contains unprocessed routing information that has been received from peers. Loc-RIB contains the routes that have been selected by the local BGP speaker's decision process, and the Adj-RIB-Out contains the routes for advertisement to specific peers by means of UPDATE messages.

Route Reflector Versus Route Server Versus Route Collector

Just for clarity, and to make sure you are on the same page as we are in this book, let's define route reflector, route server, and route collector. The major distinction between route reflectors and route servers lies in the IBGP semantics of route reflectors versus the EBGP semantics required for route servers. The route collector is a unique case.

Route Reflector

A route reflector is often used to eliminate the need for a full-mesh of sessions among IBGP speakers. The route reflector must know when to reflect a peer's announcement to another peer, in order to preserve IBGP semantics. The route reflector sends the best path in its Loc-RIB to all clients (except the one it learned the routes from). The route reflector will not typically modify the attributes, unless told to by setting the `next-hop-self` knob or local configured policy, but that is the default behavior for IBGP.

Route Server

A route server plays a similar role, transparently (attributes are not changed), but in this case for EBGP speakers, it must be able to suppress its own (possibly private) ASN from being prepended to the advertised routes.

It also maintains a RIB unique to each of its clients (per client Loc-Rib), whose policy specific to that client can be applied. The client gets its updates from that RIB, not from the global Loc-RIB.

Route Collector

The stranger in our midst is the route collector, as it does not forward any packets and also does not announce any prefixes to anyone. Its purpose, as its name more or less reveals, is to collect routing information. By doing so, the route collector and its accessory tools act like a looking glass to provide (in most cases) a public view of the 'routing information' known at 'a point in the network'. In an IXP context, the information provided by a route collector would provide useful information for:

- IXP members to check functionality of BGP filters
- Prospective members to evaluate the value of joining the IXP
- The operations community for troubleshooting purposes.

Going forward, we cover route servers in this book.

Route Server Attribute Transparency

As a route server primarily performs a brokering service, modification of attributes could cause route server clients to alter their BGP decision process for received prefix reachability information, thereby changing the intended routing policies of exchange participants. (See the Juniper TechLibrary for more detail: https://www.juniper.net/documentation/en_US/Junos/topics/reference/general/routing-protocols-address-representation.html.)

Contrary to the ordinary EBGP route handling rules, route servers do not update well-known BGP attributes by default, (unless explicitly configured) (https://ftp.apnic.net/apnic/training/eLearningHandouts/2017/20171025/eROU04_BGP_Attributes.pdf) and received from one of the server clients before redistributing them to other clients. Unless enforced by local IXP operator configuration, optional recognized and unrecognized BGP attributes, whether transitive or non-transitive, are also not updated by the route server and are passed on to other route-server clients. The well-known and optional attributes include:

- Next_Hop attribute
- AS_Path attribute
- Multi_Exit_Descriminator
- Communities

Path Hiding Mitigation in Route Server Deployments

In the traditional bilateral interconnection model, per-client policy control to a third-party exchange participant is accomplished either by not engaging in a bilateral interconnection with that participant or by implementing outbound filtering on the BGP session towards that participant. However, in a multilateral interconnection environment, only the route server can perform outbound filtering in the direction of the route server client; route server clients depend on the route server to perform their outbound filtering for them.

Assuming the default BGP decision process is followed, when the same prefix is advertised to a route server from multiple route server clients, the route server will select a single path for propagation to all connected clients. If, however, the route server has been configured to filter the calculated best path from reaching a particular route server client, then that client will not receive a path for that prefix, although alternate paths received by the route server might have been policy compliant for that client. This phenomenon is referred to as *path hiding*.

Using the example illustrated in Figure 1.5, four customer routers, depicted by C [1-4] in four different BGP autonomous systems (AS) exchange routes. C1 in AS64496 does not directly exchange prefix information with either C2 in AS644967, or C3 in AS64498 at the IXP, but only interconnects with C4 in AS64499. The lines between AS64496, AS64497, AS64498, and AS64499 represent interconnection relationships, whether via direct (bilateral) EBGP sessions or using the route server (multilateral).

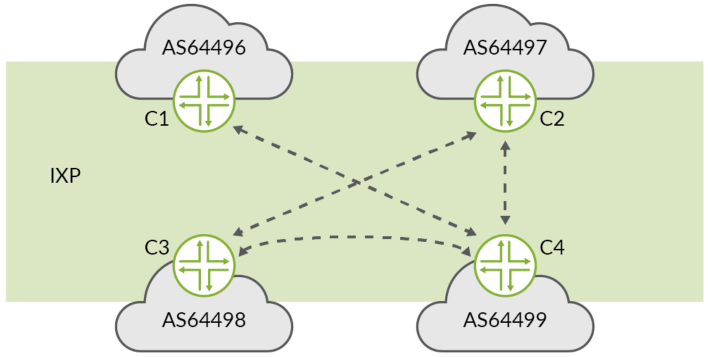


Figure 1.5 *Path Hiding at an IXP*

Let's say a prefix is advertised to the route servers from both AS64497 and AS64499, and the route via AS64497 was preferred according to the BGP decision process on the route server. All would be fine and the prefix is reachable. The exception occurs when AS64497's policy prevented the route server from sending the path to AS64496, so AS64496 would never receive a path to this prefix, even though the route server also received a valid alternative path via AS64499. This happens because the BGP decision process is performed only once on the route server for all clients.

While there are several options available to mitigate path hiding (https://tools.ietf.org/html/rfc7947#section-2.3.2) in route server environments, Junos OS employs multiple route server-client RIBs (see Figure 1.6 and https://tools.ietf.org/html/rfc7947#section-2.3.2.1). The Juniper route server BGP implementation performs the per-client best path calculation for each set of paths to a prefix, using per-client Loc-RIBs, with path filtering implemented between the Adj-RIB-In and the per-client Loc-RIB. More details on this are provided later in this book.

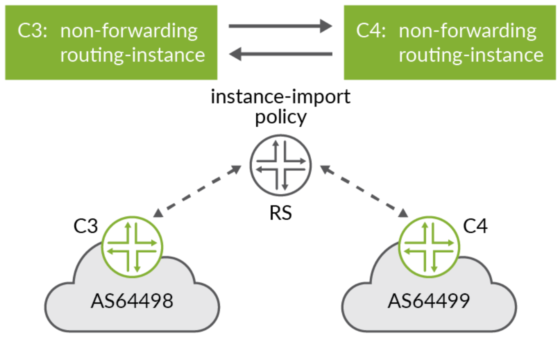

C3: non-forwarding routing-instance
C4: non-forwarding routing-instance
instance-import policy
RS
C3
C4
AS64498
AS64499

Figure 1.6 *Junos Loc-RIB Implementation*

Architecture of a Route Server Deployment

This section provides some conceptual ideas on how to set up and operate route servers in an IXP environment.

Per Route-Server Client Policy Using BGP Communities

Policy control is typically handled through the use of BGP communities. Prefixes sent to the route server are tagged with specific standard BGP communities (https://tools.ietf.org/html/rfc1997), extended communities (https://tools.ietf.org/html/rfc4360), or large communities (https://tools.ietf.org/html/rfc8092) attributes, based on predefined values agreed upon between the IXP and all IXP members. Currently there is no mutually agreed upon standard across IXPs for community usage, although some work has been done to define a standard, for example: https://tools.ietf.org/html/draft-adkp-grow-ixpcommunities-00.

In this case we use standard communities for the purpose of explaining. The usage of extended or large communities can have advantages.

BGP routes may be propagated to all other clients, a subset of clients, or no clients, depending on the values of the communities. This mechanism allows route server clients to instruct the route server to implement per-client export routing policies. As an example, IXP members may tag their routes with those shown in Table 1.1 to control policy via the route server.

Table 1.1 Example Standard BGP Communities for Controlling Per-Client Policy

Policy Description	BGP Community
Block announcement of a route to a certain peer	0:<peer-as>
Announcement of a route to a certain peer	<rs-as>:<peer-as>
Block announcement of a route to all peers	0:<rs-as>
Announcement of a route to all peers	<rs-as>:<rs-as>

Redundancy

The purpose of an IXP route server implementation is to provide a reliable reachability brokerage service, therefore IXP operators generally deploy multiple route servers (see Figure 1.7).

It may be a good idea to also advertise routes between route servers to ensure reachability even if, or when, BGP sessions are down for specific IXP member routers on specific route servers, but not on others. While this event is likely to be quite rare, it's worth carefully considering it when offering a route server-based service.

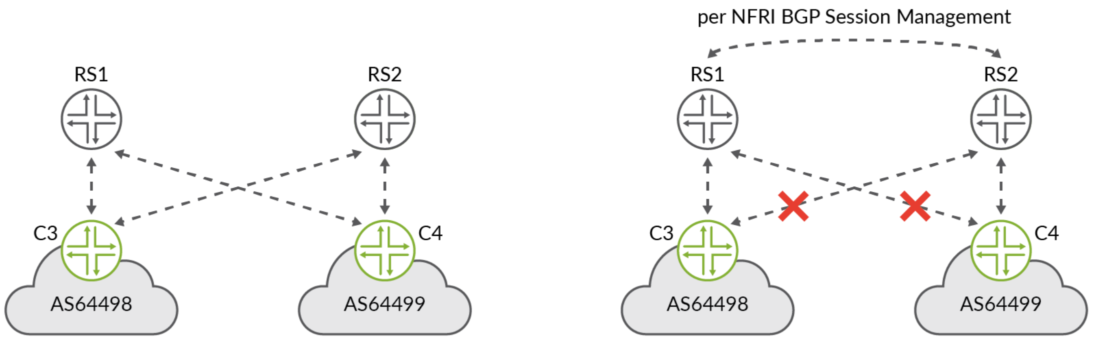


Figure 1.7 *Redundant Route Server Deployment*

However, redundant route server deployments of this style may result in a bit more complexity in terms of session management between route servers as well as instance-import/export policies associated with Junos OS path mitigation techniques. In Figure 1.7, the right side depicts this extra complexity in the form of per Non-Forwarding Routing Instance (NFRI) BGP sessions and policy. Therefore, this has to be carefully evaluated. Most IXPs operate a set of two independent route servers as this seems to result in the best balance between redundancy and complexity.

Route Server Security Considerations

The following basic EBGP best practices are for your security considerations.

Generalized TTL Security Mechanism (RFC3682)

GTSM is designed to protect a router's control plane from CPU-utilization based attacks. GTSM is based on the fact that the vast majority of protocol peerings are established between routers that are adjacent, as is the case of EBGP peers on an IX LAN. Since TTL spoofing is considered nearly impossible, a mechanism based on an expected TTL value can provide a simple and reasonably robust defense from infrastructure attacks based on forged protocol packets from outside the network.

Session Authentication (RFC2385)

A typical IXP peering LAN consists of multiple switches forming one large Layer 2 fabric. The downside of this architecture is that it is fairly easy to ‘hijack’ another member’s BGP session by spoofing MAC and/or IP addresses. It’s good practice for an IXP to deploy filters on access ports, for example, to restrict a member to only use a specific MAC address.

Securing BGP Sessions (MD5/TCP-AO)

Securing the BGP session itself is considered an additional layer of security, making sure you are peering with who you think you are peering with.

MD5 is a TCP extension to enhance security for BGP. It defines a new TCP option for carrying an MD5 (https://tools.ietf.org/html/rfc1321) digest in a TCP segment. This digest acts like a signature for that segment, incorporating information known only to the connection end points (peers). Since BGP uses TCP as its transport, using MD5 significantly reduces the danger from certain security attacks on BGP.

However, in 1996 a flaw was found in the design of MD5, and in 2004 it was shown that MD5 is not collision resistant. So MD5 is not considered suitable, as it is deprecated, and insecure (https://en.wikipedia.org/wiki/MD5#Security), but it is still widely used. Today there are better alternatives available, for example, the TCP Authentication Option (RFC5925; https://tools.ietf.org/html/rfc5925). Unfortunately this has not been implemented by many network vendors, until now.

Maximum Prefix Limits

Setting a limit on the number of prefixes accepted from a peer is one of the simplest things that can be done to protect the route server from being intentionally or unintentionally overloaded due to routing or policy mistakes by IX members. The purpose of this limit is to serve as a final fail-safe. If an import policy fails, bringing down the EBGP session, it will send an alert that something “incorrect” has happened.

There are several schools of thought pertaining to defining what the “maximum” number of prefixes should be. Is it the maximum number of received prefixes, before import policies and/or BGP `best-path` is performed, or is it the maximum number of accepted prefixes after import policy is applied and a BGP `best-path` is calculated?

Further, there are various recommendations for what the maximum value should be, for example, ten percent of the number of prefixes you’d expect from an IX member router? The problem is that when these hard limits are set, it is easy to forget they are in place, thus inadvertently causing your session(s) to be reset if a sudden jump in prefixes is encountered.

Therefore, the max value should be high enough to prevent accidentally tripping it, but also low enough so as not to blindly accept and possibly accidentally accept a full routing table. A few suggestions may be:

1. Multiply the number of routes expected and multiply by 10.
2. Use a logarithmic scale to derive an appropriate limit.
3. Leverage an external application or controller to monitor, learn, and modify per session maximums. This solution will be looked at in more detail in Chapter 5.

Table 1.2 Maximum Prefix Value Example

Expected # of Routes	Simple (x10) Maximum Values	Log(n) Maximum Values
10	100	100
1,000	10,000	3,000
50,000	500,000	234,500

Applying outbound maximum prefix filters isn't currently a widely used technique, however, it could help prevent you from leaking "a full table" due to "fat fingers."

As with inbound filtering, the maximum prefix is considered a "final fail-safe" in case 'the other side' makes a mistake, so you could consider this as preventing you from hurting others.

IETF is currently working on a draft called "BGP Maximum Prefix Limits" in in order to standardize pre/post in/outbound maximum prefix limits: https://datatracker.ietf.org/doc/draft-sa-grow-maxprefix/.

Maximum prefix limits, among other parameters, is something you could typically want to automate because it can change rapidly and frequently. For example, when one member buys another member or connects a new customer to its network, you have plenty of filters to update. It's good practice as a network operating in the Default Free Zone (DFZ) (see: https://en.wikipedia.org/wiki/Default-free_zone) to keep an up-to-date PeeringDB (https://www.peeringdb.com/) profile. Many networks already use PeeringDB information to automatically build their filters. An example to get you started using Python to automate parts of the peering process can be found here: https://github.com/coloclue/kees and at https://github.com/coloclue/kees/blob/master/peering_filters.

Default EBGP Route Propagation Behavior Without Policies (RFC8212)

By default, many BGP speakers (routers) advertise and accept any and all route announcements between their neighbors. This dates back to the early days of the Internet, when operators were permissive in sending routing information to allow all networks to reach each other. As the Internet has become more densely interconnected, the likelihood of a misbehaving BGP speaker poses significant risks to the global routing table and also to the Internet.

RFC8212 (https://tools.ietf.org/html/rfc8212) addresses this situation by requiring the explicit configuration of both BGP import and export policies for any EBGP session such as customers or peers. BGP speakers following this specification do not use or send routes on EBGP sessions unless specifically configured to do so. In other words, there is a policy in place to explicitly advertise routing information to a neighbor.

Per Route Server Client Prefix Validation

Many IXPs validate prefixes at ingress on all route servers. The validation is based on Internet Routing Registry (IRR) or Route Object Authorization (ROA) object presence. A list of valid origin ASNs and valid prefixes based on route objects is constructed in the form of route-filter-lists or prefix-lists along with aspath-lists. More specific announcements of valid routes are often rejected due to missing ROAs. A valid AS-SET in the IXP members PeeringDB record is searched. If no valid AS-SET is found, often only the member's ASN is used.

Table 1.3 below shows an example of ingress standard community-based tagging based on the results of ingress validation.

Table 1.3 Example of Ingress Standard Community-Based Tagging

Policy Description	BGP Community
Prefix is present in an AS’s announced AS/AS-SET	<rs-as>:650010
Prefix is not present in an AS’s announced AS/AS-SET	<rs-as>:650011
Prefix has valid Origin AS in AS-SET	<rs-as>:650012
Prefix has no valid Origin AS in AS-SET	<rs-as>:650013

The prefix validation often occurs, and IXP members can check the communities being set to their prefixes and see the results of the validation checks through a route server or a route collector looking glass. At egress, filtered prefixes that have failed the validation are rejected. Some IXPs offer members who prefer to receive an unfiltered set of prefixes to opt out. This is not advised for obvious reasons; there's no good reason to keep invalid routing information in your routing tables.

Origin Validation for BGP Using RPKI

A substantial part of the route advertisements seen in the global routing table are invalid, or no valid Route Origin Authorizations (ROAs) (https://www.ripe.net/manage-ips-and-asns/resource-management/certification/resource-certification-roa-management#---route-origin-authorisations--roas-) could be found, as can be seen in this NIST RPKI monitor: https://rpki-monitor.antd.nist.gov/. While writing this book in early June 2019, 0.74% routes were invalid and for 86.32% of the global routing table ROAs were not found. Obviously, those numbers need to go down as fast as possible!

The most common routing error is the accidental route leak due to policy error or *mis-origination* of a prefix (fat fingers), meaning someone unintentionally announces an IP prefix that they are not the holder of, or they advertise a more specific route without a valid routing object in an RIR database. The latter would cause BGP routers to declare a certain route "better" and to prefer it over the correct route. This is probably not intended as it does not lead to the network of the rightful owner of the IP space. As a (partial) answer to this problem, *origin validation*, using resource public key infrastructure (RPKI), offers BGP origin validation. The question it tries to answer is: "*Is this particular route announcement authorized by the legitimate holder of the address space?*"

RPKI allows operators to create cryptographically signed statements about their route announcements. These statements are called route origin authorization (ROAs). A ROA states which AS is authorized to originate (advertise) a certain IP prefix. In addition, it can determine the maximum length of the prefix that the AS is authorized to advertise. Based on this information, other networks that have deployed origin validation on their routers can then validate if the advertisements they receive are valid or invalid, and use that information to make routing decisions.

In addition to connecting networks, an IXP has a responsibility to keep the Internet safe and stable. From that perspective, deploying origin validation on your route servers should be a no-brainer, although in which direction (inbound or outbound) you would want to deploy RPKI requires some thought. Ideally, dropping malicious advertisements happens when they enter your network, or in this case, your route server. However, this will make troubleshooting harder for you as you will not be able to see what routes are advertised to the route server.

If your customer wants to use your looking glass to check if the route server has received the prefix, for example, they would not get a result as it's filtered between Adj-RIB-in and Loc-RIB.

So in practice you might want to accept the advertisements to enter Loc-RIB, and filter them when entering Adj-RIB-out, in order to prevent your route server from advertising the invalid routes to your customers.

NOTE Covering RPKI in full detail is beyond the scope of this book. It is covered in great detail in the book *Day One: Deploying BGP Routing Security* at https://www.juniper.net/us/en/training/jnbooks/day-one/deploying-bgp-routing-security/.

MORE? How to configure your Junos OS router or route server to perform origin validation is described here: https://www.juniper.net/documentation/en_US/Junos/topics/topic-map/bgp-origin-as-validation.html.

Policy Implementation Considerations

Before jumping into the policy implementation details of a route server, it's worth reviewing a few nascent BGP implementation considerations. Any BGP speaker receiving routing updates from other peers processes the information for local use and then advertises selected routes to different peers based on predefined policies. In order for BGP to be able to perform this function, it stores this information in a special type of database called the *BGP Routing Information Base*.

A BGP Routing Information Base consists of three parts:

1. The *Adj-RIB-In*: BGP RIB-In stores BGP routing information received from different peers. The stored information is used as an input to the BGP decision process. In other words, this is the information received from peers before applying any attribute modifications or route filtering.
2. The *Local RIB*: The local routing information base stores the post policy information after processing the RIB-In information. These are the routes that are used locally after applying BGP policies and the decision process.
3. The *Adj-RIBs-Out*: This one stores the routing information that was selected by the local BGP router to advertise to its peers through BGP update messages.

NOTE Chapter 3 provides detailed CLI examples for monitoring each of the three BGP RIB parts.

This basic routing information flow, from client to route server and back to client, is depicted in Figure 1.8. The figure also illustrates where specific policies could be applied to administer the various IXP and general BGP policies just described.

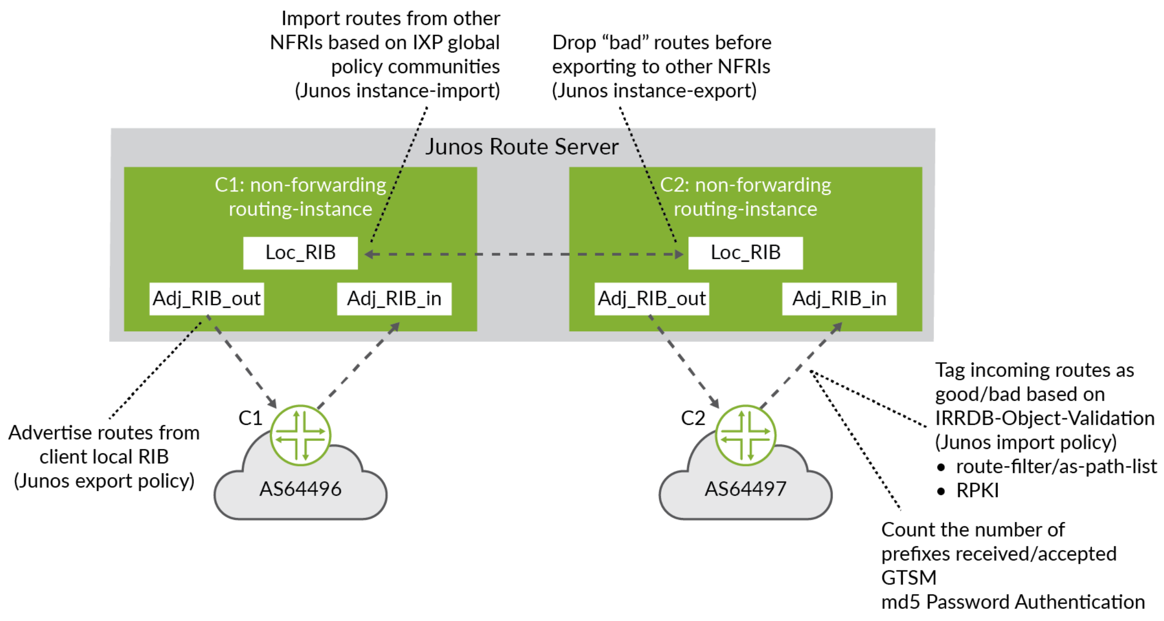

Import routes from other NFRIs based on IXP global policy communities (Junos instance-import)
Drop “bad” routes before exporting to other NFRIs (Junos instance-export)
Junos Route Server
C1: non-forwarding routing-instance
Loc_RIB
Adj_RIB_out
Adj_RIB_in
C2: non-forwarding routing-instance
Loc_RIB
Adj_RIB_out
Adj_RIB_in
C1
AS64496
C2
AS64497
Advertise routes from client local RIB (Junos export policy)
Tag incoming routes as good/bad based on IRRDB-Object-Validation (Junos import policy)
• route-filter/as-path-list
• RPKI
Count the number of prefixes received/accepted
GTSM
md5 Password Authentication

Figure 1.8 Route Server Configuration Example Workflow

The databases described here are not to be confused with the routing table as these are only the tables used by the BGP process and never by the router for packet forwarding. Only the set of routes that exist in the Local-RIB are installed in the routing table based on a criterion specified by the local BGP speaker (depending on vendor implementation and preference of routing protocols).

Chapter 2

Route Server Implementation

Traditionally you would expect a physical box, for example a MX Series router running Junos 17.4R1, to be used for the route server role in a network. This is obviously possible, but you could just as well be running virtual appliances, such as vMX or vRR. However, since Juniper *containerized* its routing protocol daemon (rpd), it is now also possible to run typical control plane functions like a route server off-box as a container. What this means, how it works, and the advantages or disadvantages of either of these options are all covered in this chapter.

Junos OS Route Server Platforms

Any Juniper platform running an appropriate version of Junos can function as a route server. However, because route server deployment may maintain a local RIB for each EBGP client, as is often the case, BGP tends to require slightly more memory than a normal BGP deployment. As a result, the memory capacity of a normal router may not provide a scalable solution.

Figure 2.1 shows a block diagram of a typical Juniper MX Series router.

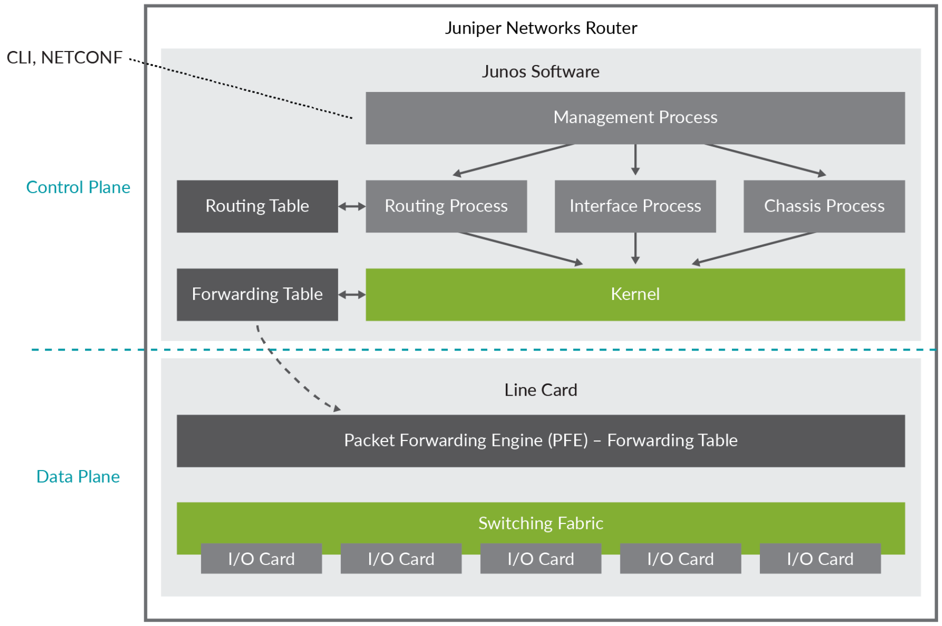


Figure 2.1 *Juniper MX Series as a Route Server*

The memory capacity of a typical router is often dimensioned to meet the target deployment “role” of that router. For example, a Juniper PTX10000 Series serving as a core router doesn’t have the same feature set – and thus memory requirements – as a route server, not to mention the additional hardware in the form of line cards and packet forwarding engines that are necessary for a core router.

Junos OS Virtual Route Reflector (vRR)

The Junos OS virtual Route Reflector (vRR) allows the deployment of the Junos OS control plane using a general-purpose VM that can run on a 64-bit Intel-based server like the Juniper NFX platform or an appliance like the Juniper JRR200. A vRR on an Intel-based server or appliance works the same as a route reflector on a router running BGP, providing a scalable and low-cost alternative to a dedicated hardware platform. The vRR has the following benefits:

- Scalability: Scalability improvements, depending on the server core hardware (CPU and memory) on which the vRR runs.
- Faster and more flexible deployment: vRR running on an Intel server, using open source tools, which reduces typical router maintenance.

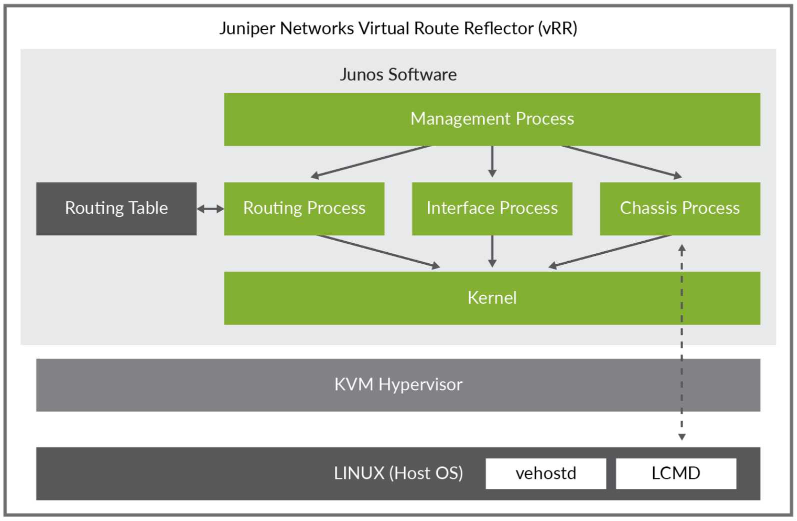


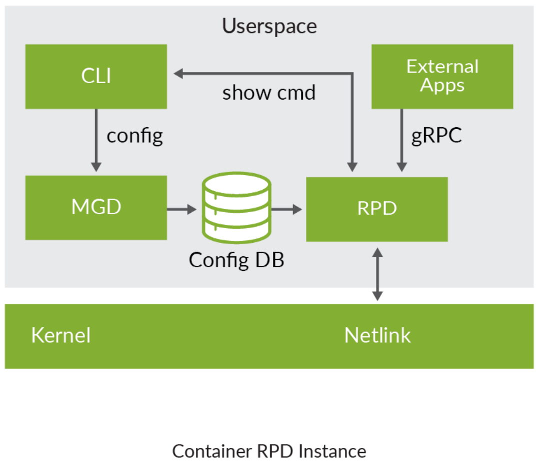
Figure 2.2 Juniper Virtual Route Reflector (vRR)

Deploying a route server using a vRR platform allows the IXP to dimension the server's CPU and memory to that of the target deployment. Since modern server platforms have far more compute and storage capacity than a traditional Routing Engine, it ensures that the route server can both scale and perform well beyond a dedicated router.

Junos OS Containerized RPD (cRPD)

As depicted in the previous two diagrams the rpd is a core component of the Junos OS, and it is responsible for running various routing protocols (OSPF, ISIS, RIP, BGP, MPLS, etc.) to learn and distribute route state. A *containerized rpd* (cRPD) is designed to facilitate rpd as a standalone module that is decoupled from the base Junos OS and able to run in Linux-based environments. These environments can be as diverse as host or server systems, VMs, containers, and network devices with separate physical or virtual forwarding planes.

Additional options for route-server deployments are provided by cRPD, with the simplest being a model much like using a vRR. However, cRPD provides many other avenues to be explored by spawning many instances of rpd behind a Docker bridge. Thus, a single route server instance is presented to the IXP clients, but behind that Docker bridge may be a cluster of rpd instances. This potentially leads to possibly greater scale, new high availability models, various software versions, and in general, a new way to think about what a route server can be. The basic cRPD instance and a high-level view of what a scaled-out route server may look like is shown in Figure 2.3.



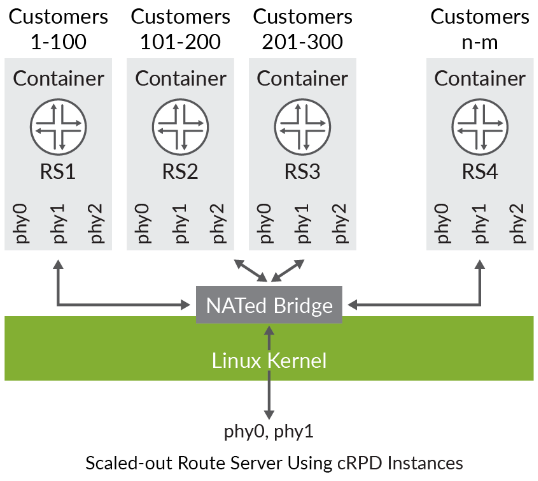


Figure 2.3 *cRPD and cRPD as a Scaled-out Route Server*

While specific procedures will vary, you can import a Junos OS docker image into your environment; create a container and bring cRPD up using these steps.

Step 1: Download cRPD from https://support.juniper.net/support/downloads/:

```
docker load –i crpd:19.2R1.tar.gz
```

Step 2: Create persistent storage volume for config and logs (default size: 10GB):

```
docker volume create crpd1-config
docker volume create crpd1-log
```

Step 3: Create and launch container:

```
    docker run --rm --detach --name crpd1 -h crpd1 --privileged  -v crpd1-config:/config -v crpd1-
log:/var/log -it crpd:19.2R1
```

NOTE It is important to launch cRPD in privileged mode. Failure to do so results in the daemons not starting.

To limit the amount of memory and CPU allocated to a cRPD container (for example, 256 MB and 1 CPU):

```
   docker run --rm --detach --name crpd2 -h crpd2 --privileged -v crpd1-etc:/etc -v crpd1-config:/
config -v crpd1-log:/var/log \
                -m 256MB --memory-swap=256MB –ncpus=1 -it crpd:19.2R1
```

NOTE 256MB is the lowest amount of memory needed to support cRPD. For a route-server scenario, a much higher amount of memory is recommended.

To launch a cRPD container in host networking mode (shares networking name space with host, default is separate name space):

```
    docker run --rm --detach --name crpd1 -h crpd1 --privileged  --net=host -v crpd1-config:/
config -v crpd1-varlog:/var/log \
                     -it crpd:19.2R1
```

Basic Management of cRPD

Checking the container status:

```
    docker ps          Lists currently running containers
    docker stats       Displays per container resource utilization information
```

Access cRPD CLI: Launches JUNOS CLI:

```
    docker exec -it crpd1 cli
```

Access cRPD bash shell:

```
    docker exec -it crpd1 bash
```

Pause/Resume/Stop/Destroy the container:

```
  docker [ pause | resume |stop | rm ] crpd1
```

See Appendix C for specific cRPD configuration examples and considerations.

Route Server Configuration

The following configuration, monitoring, and troubleshooting examples are based on the example IXP network shown in Figure 2.4.

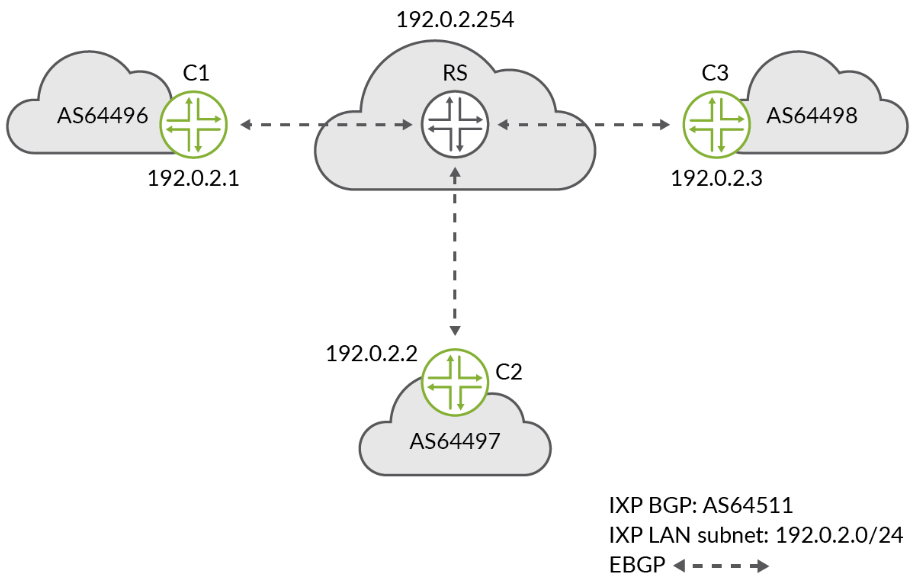


Figure 2.4 Example IXP Network

Basic Junos Route-Server Client Configuration

The Junos route server supports EBGP transparency by modifying the normal BGP route propagation such that neither transitive nor non-transitive attributes are stripped or modified while propagating routes. Changes to normal EBGP behavior are controlled by the CLI configuration `route-server-client`:

```
protocols {
    bgp {
        group C1 {
            type external;
            route-server-client;
            family inet {
                unicast;
            }
            neighbor 192.0.2.1 {
                peer-as 64496;
```

Junos Route Server-Client Configuration using a Non-Forwarding Routing Instance

A route-server client-specific RIB is a distinct *view* of a BGP Loc-RIB that can contain different routes for the same destination than other *views*. Route-server clients, via their peer groups, may associate with one individual client-specific view or a shared common RIB.

In order to provide the ability to advertise different routes to different clients for the same destination, it is conceptually necessary to allow for multiple instances of the BGP path selection to occur for the same destination but in different client/group contexts.

Junos OS implements flexible policy control with per client/group path selection, using non-forwarding routing instances to provide multiple instances of the BGP pipeline, including BGP path selection, Loc-RIB, and policy. A Junos route server will be configured to group route-server clients within BGP groups configured within separate non-forwarding routing instances. This approach leverages the fact that BGP running within a routing instance does path selection and has a RIB that is separate from BGP running in other routing instances:

```
routing-options {
    router-id 192.0.2.254;
    autonomous-system 64511;
}
protocols {
    bgp {
        group C1 {
            type external;
            route-server-client;
            family inet {
                unicast;
            }
            neighbor 192.0.2.1 {
                peer-as 64496;
            }
        }
    }
}
routing-instances {
    C1 {
        instance-type no-forwarding;
        routing-options {
            router-id 192.0.2.254;
            instance-import [ <instance-import-policy ( s ) > ];
```

Basic Route-Server Client Configuration

The following CLI example shows a basic IXP member client router's configuration. As you can see, it's quite simple, compared to the configuration of the route server, since the client, in the most basic sense, needs only to tag the routes they want to advertise at the IXP with the required communities, and the route server does the rest:

```
routing-options {
    router-id 192.0.2.1;
    autonomous-system 64496;
}
protocols {
    bgp {
        group ebgp_inet {
            family inet {
                unicast;
            }
            export BGP-export;
            neighbor 192.0.2.254 {
                peer-as 64511;
```

In this example, let's assume that C1's policy is open and allows its prefixes to be advertised to all IXP members, thus it attaches the standard BGP community 64496:123 to all of its advertised prefixes:

```
policy-options {
    policy-statement BGP-export {
        term 0 {
            from {
                route-filter 203.0.113.0/24 orlonger;
            }
            then {
                community set as64496_comms;
                accept;
            }
        }
    }
    community as64496_comms members 64496:123;
```

Whereas C2's peering policies dictate that their prefixes only be advertised to C1, but not to anyone else. Their prefixes are in turn advertised with BGP standard community 64497:64496 indicating to the route server to restrict instance-imports to only C1's routing-instance:

```
policy-options {
    policy-statement BGP-export {
        term 0 {
            from {
                route-filter 198.51.100.0/24 orlonger;
            }
            then {
                community set as64497_comms;
                accept;
            }
        }
    }
    community as64497_comms members 64497:64496;
}
```

Default EBGP Route Propagation Behavior Without Policies (RFC8212)

Junos OS does not yet natively support RFC8212 (https://tools.ietf.org/html/rfc8212), therefore a SLAX script is available and should be used in order to accomplish this behavior.

Rfc8212.slax, is a strict implementation that replaces the absence of RFC8212 implementation with a `deny-all` statement if no export policy is present.

A loose form of the policy can be found at: https://github.com/packetsource/rfc8212-Junos .

```
Copy rfc8212.slax to /var/db/scripts/commit
```

Apply the following configuration change:

```
system {
    scripts {
        commit {
            file rfc8212.slax;
```

Generalized TTL Security Mechanism (RFC3682)

The Generalized TTL Security Mechanism (GTSM) is designed to protect a router's IP-based control plane from CPU utilization-based attacks. GTSM is based on the fact that the vast majority of protocol peerings are established between adjacent routers, as is the case of EBGP peers on an IX LAN. Since TTL spoofing is considered nearly impossible, a mechanism based on an expected TTL value can provide a simple and reasonably robust defense from infrastructure attacks based on forged protocol packets from outside the network:

```
interfaces {
    em0 {
        unit 0 {
            family inet {
                filter {
                    input ttl-security;
                }
            }
        }
    }
}
protocols {
    bgp {
        group C1 {
            type external;
            route-server-client;
            family inet {
                unicast;
            }
            neighbor 192.0.2.1 {
                peer-as 64496;
            }
        }
    }
}
policy-options {
    prefix-list EBGP_PEERS {
        apply-path "routing-instances <*> protocols bgp group <*> neighbor <*>";
    }
}
firewall {
    filter ttl-security {
        term gtsm {
            from {
                source-prefix-list {
                    EBGP_PEERS;
                }
                protocol tcp;
                ttl-except 225;
                port 179;
            }
            then {
                discard;
            }
        }
        term else {
            then accept;
```

Maximum Prefix Limits

The BGP `prefix-limit` feature allows you to control how many prefixes can be received from a BGP neighbor. The prefix-limit feature is useful to ensure a client router does not accept more than a previously agreed upon number of prefixes from a route server. This prevents intentionally or unintentionally overloading a router server by an IXP member:

```
routing-instances {
    C1 {
        protocols {
            bgp {
                group as64496 {
                    family inet {
                        unicast {
                            prefix-limit {
                                maximum 500;
                                teardown idle-timeout 30;
```

Alternatively, the route server can be configured to limit the number of prefixes that are accepted from an IXP member. This may or may not be useful, depending on how the input prefix validation is handled by the IXP. For example, if invalid routes are discarded, then the number of accepted prefixes may differ from the number of advertised prefixes:

```
routing-instances {
    C1 {
        protocols {
            bgp {
                group as64496 {
                    family inet {
                        unicast {
                            accepted-prefix-limit {
                                maximum 500;
                                teardown 80 idle-timeout 30;
```

Local RIB Import/Export Policy Configuration Examples

To propagate routes between route server clients, routes need to be leaked between the RIBs of the routing-instances based on configured policies and/or well-defined community values. Configuring a full mesh of route leaking between n client-specific routing instances using `instance-import` and the `instance` qualifier requires n table-specific policies, each with n-1 policy terms, explicitly naming the routing instances from which to leak. Thus, the policy configuration explodes on the order of O (n^2).

As a configuration convenience, an `instance-any` policy qualifier is provided for use with `instance-import` policies. The `instance-any` qualifier has the functionality of leaking routes from all other routing instances into the instance in which `instance-import` is configured. The `instance-import` may have other qualifying terms to implement further filtering to match the IXP global community policies.

Furthermore, the example routing policies described next will follow the workflow illustrated in Figure 2.5 and described in Chapter 1. The workflow follows routes announced from client C1 to the route server and then onto client C2.

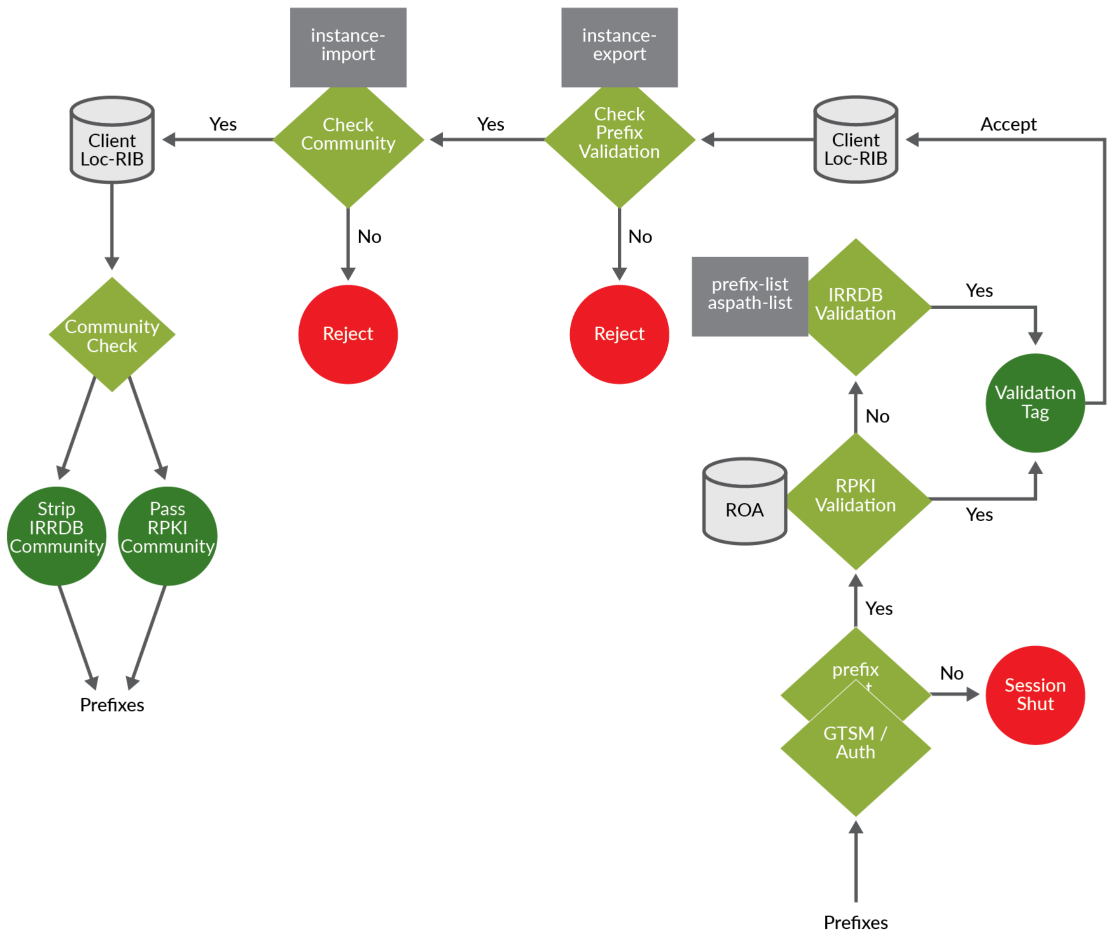


Figure 2.5 Route-Server Configuration Example Workflow

Example Instance Import Policies

Let's first look at how to use the `instance-any` policy qualifier along with a standard community matching filter to import routes from all routing instances that have an open routing policy.

You can see how the policy term first uses the `instance-any` qualifier and then matches community 64511:111 which, for this example, represents the IXP global community for an open policy. Then that policy is applied at the `routing-instance instance-import` attachment point:

```
policy-options {
    policy-statement from-any {
        term from-any-instance {
            from {
                instance-any;
                community announce-to-all;
            }
            then accept;
        }
    }
    community announce-to-all members 64511:111;
}
routing-instances {
    C1 {
        routing-options {
            instance-import from-any;
```

As alluded to at the beginning of this section, an alternative way to configure a subset of instances is to use the `instance-list` identifier. You can see next that instead of using the `instance-any` qualifier we have listed each individual client routing instance along with a community value of 0:100, which is the IXP global community, to block routing advertisements to specific route-server clients:

```
policy-options {
    policy-statement block-to-as64496 {
        term from-any-inst {
            from {
                instance-list [ C2 C3 C4 C5 ];
                community block-to-as64496;
            }
            then reject;
        }
    }
    community block-to-all members 64511:64496;
}
routing-instances {
    C1 {
        routing-options {
            instance-import block-to-as64496;
```

Example Import Policy

This example policy accepts and tags prefixes with a valid/invalid community value based on IRR object validation as described in the previous prefix validation section. Several examples using third-party tool integration, as described in a later section, may be leveraged to automatically build the `route-filter-list` and `as-path-group` lists. The example policy here adds the community tags to be used in later instance import/export policies:

```
policy-options {
    route-filter-list as64496-prefixes {
        198.51.100.0/24 exact;
        203.0.113.0/24 exact;
    }
    policy-statement irr-object-validation {
        term good-prefix {
            from {
                route-filter-list as64496-prefixes;
            }
            then {
                community add good-prefix;
                accept;
            }
        }
        term bad-prefix {
            then {
                community add bad-prefix;
                next term;
            }
        }
        term good-origin {
            from as-path-group as64496-aspaths;
            then {
                community add good-origin;
                next term;
            }
        }
        term bad-origin {
            then {
                community add bad-origin;
                accept;
            }
        }
    }
    community bad-origin members 64511:644963;
    community bad-prefix members 64511:644961;
    community good-origin members 64511:644962;
    community good-prefix members 64511:644960;
    as-path-group as100-aspaths {
        as-path as64496-1 ".*(2|109|543|714|1221)";
        as-path as64496-2 ".*(1248|1290|1766|1828|1851)";
        as-path as64496-3 ".*(1873|2047|2161|2198|2559)";
    }
}
routing-instances {
    C1 {
        instance-type no-forwarding;
        protocols {
            bgp {
                group C1 {
                    neighbor 192.0.2.1 {
                        import irr-object-validation;
                        peer-as 64496;
```

Example Origin Validation for BGP (RPKI)

An alternative to writing and maintaining explicit `as-path-group` and `route-filter-list` lists is to use RPKI, as described earlier in this book. The following policy checks the RPKI validation database and marks destinations with the returned validation community; `valid`, `invalid`, or `unknown`:

```
policy-options {
    policy-statement validation {
        term valid {
            from {
                protocol bgp;
                validation-database valid;
            }
            then {
                validation-state valid;
                community add origin-validation-state-valid;
                accept;
            }
        }
        term invalid {
            from {
                protocol bgp;
                validation-database invalid;
            }
            then {
                validation-state invalid;
                community add origin-validation-state-invalid;
                accept;
            }
        }
        term unknown {
            from {
                protocol bgp;
                validation-database unknown;
            }
            then {
                validation-state unknown;
                community add origin-validation-state-unknown;
                accept;
            }
        }
    }
    community origin-validation-state-invalid members 0x4300:0.0.0.0:2;
    community origin-validation-state-unknown members 0x4300:0.0.0.0:1;
    community origin-validation-state-valid members 0x4300:0.0.0.0:0;
```

The format of the communities shown here result from RFC8097 Section 2, where the value of the high-order octet of the extended type field is 0x43, which indicates it is non-transitive. The value of the low-order octet of the extended type field, as assigned by IANA, is 0x00. And the last octet of the extended community is an unsigned integer that gives the route's validation state with the following values:

- 0 = Lookup result = `valid`
- 1 = Lookup result = `not found`
- 2 = Lookup result = `invalid`

Example Routing Instance Export Policies

This next instance export policy example rejects any prefix that has been marked as invalid by either the RPKI database lookup, a `route-filter-list` miss, or an `as-path-group` miss. All other prefixes allowed are then checked by instance-import policies:

```
policy-options {
    policy-statement validation-failure {
        term bad-prefix {
            from community bad-prefix;
            then reject;
        }
        term bad-origin {
            from community bad-origin;
            then reject;
        }
        term rpki-invalid {
            from community origin-validation-state-invalid;
            then reject;
        }
        term accept {
            then accept;
        }
    }
}
routing-instances {
    C1 {
        routing-options {
            instance-export validation-failure;
```

Example Export Policy

Most, if not all, policy has already been implemented between import policy, instance-export policy, and instance-import policy. The only remaining policy may be to remove the prefix validation communities before sending the prefixes on to route server clients. This may also include the RPKI valid/invalid/unknown communities. The example RPKI policy shown above is a pretty strict policy in that RPKI invalid prefixes are dropped at `instance-export`. It can be the IXP policy to merely pass on the results, as a service, to the members of the IXP, thus allowing them to decide how to use the validation results.

BEST PRACTICE Never propagate 'RPKI invalids'; there is no single good reason to do so: "`invalid == reject`" is the only legitimate option. https://www.google.nl/search?q=Invalid+%3D%3D+reject.

Chapter 3

Scaling, Troubleshooting, and Monitoring Considerations

This chapter discusses various aspects of monitoring route server scale, Junos BGP components, and client-specific BGP sessions. Monitoring a route server is not unlike *normal* BGP session management, which is covered so commonly in other publications that we assume the reader is well-versed in it.

Unlike those normal BGP speakers, Junos route servers have a couple of special considerations:

- Configuration database size
- rpd memory utilization for route copies between routing instances

Monitoring the Configuration Database Size

To support large configurations, for example say a thousand route server clients resulting in more than two million lines of output or more, the default configuration database size needs to be extended and compression enabled. The following configuration stanza enables extended configuration database size but *requires* a Junos reboot:

```
system {
    configuration-database {
        extend-size;
    }
    compress-configuration-files;
```

Initial Junos Configuration Database

The Junos configuration database can be monitored with the `show system configuration database usage` command, here showing the size with a basic configuration:

```
root@rs1> show system configuration database usage
Maximum size of the database: 1305.99 MB
Current database size on disk: 1.50 MB
Actual database usage: 1.47 MB
Available database space: 1304.52 MB
```

Now let's show 1,000 route-server clients, each with their own routing instance, import policy, instance-export policy, and instance-import policy:

```
root@rs1> show system configuration database usage
Maximum size of the database: 1305.99 MB
Current database size on disk: 1221.50 MB
Actual database usage: 1221.48 MB
Available database space: 84.51 MB
```

You can see the significant database usage.

Monitoring Route Table Size

The next sample output shows summary statistics about the entries in the routing table (`show route summary` command) and the memory usage breakdown (`show task memory detail` command) for the rpd. The two commands provide a comprehensive picture of the memory utilization of the routing protocol process.

The `show route summary` command shows the number of routes in the various routing tables for each route server client. Within each routing table, all of the active, holddown, and hidden destinations and routes are summarized. Routes are in the `holddown` state prior to being declared `inactive`, and hidden routes are not used because of routing policy. Note that routes in the `holddown` and `hidden` states are still using memory because they appear in the routing table:

```
root@rs1> show route summary table C1.inet.0
Autonomous system number: 123
Router ID: 192.0.2.254

C1.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
                  BGP:     31 routes,     31 active
```

Monitoring RPD Memory Utilization

The `show task memory detail` command lists the data structures within the tasks run by rpd. Tasks are enabled depending on the router's configuration. The `Alloc Bytes` field indicates the highest amount of memory used by the data structure. The maximum allocated blocks and bytes are high water marks for a data structure. The example below looks to be output from a very healthy route server as very little memory is being allocated:

```
root@rs1> show task memory detail | match "Total bytes|----|bgp|Allocator|Name|Malloc" | except je_
task | except "Size TXP" | except Overall |except Page

  -------------------------------------------------------------------------
  ------------------------ Allocator Memory Report ------------------------
  Name                   Size Alloc DTXP    Alloc      Alloc MaxAlloc  MaxAlloc
  bgp-th-wrstage-iov    10240 12288            5      61440        7     86016
  rt_table_name_node       20    24           11        264       12       288
  bgpconf                1776  1792           12      21504       17     30464
  bgp_uio_trace_conf       20    24           36        864       36       864
  bgp_uio_group_info      820   896            4       3584        4      3584
  bgp_uio_peer_info       168   192            7       1344        7      1344
  bgp_orf_mark_t           24    28            7        196        7       196
  bgp_riblist_entry_t      12    16           12        192       12       192
  bgp_rg_list_obj           8    12            1         12        1        12
  bgp_msgbld               76    80           11        880       11       880
  bgp_io_oper             228   256            7       1792        7      1792
  bgp_nlri_sync_t         372   384            7       2688        7      2688
  bgp-rib-to-group         12    16            4         64        4        64
  bgp_mrto_hash          4096  8192            2      16384        4     32768
  bgp-rib-grp            1896  2048            2       4096        4      8192
  bgp-rib-globals          28    32            7        224        7       224
  bgp-rib-peer-group      496   512            2       1024        4      2048
  bgp-rib-peer-counter    116   128            7        896        7       896
  bgp-rib-peer            484   512            7       3584        7      3584
  bgp-rib                 136   140            7        980        7       980
  bgp ifachg notify        12    16            3         48        3        48
  bgp ifachg reg           32    36            3        108        3       108
  bgp_act_node            700   768            7       5376        7      5376
  bgp_adv_entry            24    28            6        168      136      3808
  bgp_checksum_stats_t     12    16            7        112        7       112
  bgp_tsi_t                16    20            6        120      136      2720
  bgp_addpath_params_t     12    16            7        112        7       112
  bgpb_sbits_01            20    24            4         96       10       240
  bgp_metrics_node         84    96           13       1248       19      1824
  bgpg_rtinfo_entry        16    20            6        120       65      1300
  bgp_peeras_t             16    20            1         20        3        60
  bgp_rtentry              24    28           43       1204      105      2940
  bgp_bmp_common_peer_      4     8           16        128       22       176
  bgpPeerGroup          19088 20480            4      81920        4     81920
  bgpPeer                9288 12288            7      86016        7     86016
  bgp_buffer             4100  8192            5      40960        7     57344
  bgp_cluster_t            16    20            1         20        1        20
  bgp_instance_t           88    96            5        480        6       576
  bgp_evpn_metrics         48    52           18        936       34      1768
  -------------------------------------------------------------------------
  -------------------------- Malloc Usage Report ---------------------------
  Name                      Allocs     Bytes MaxAllocs  MaxBytes  FuncCalls
  BGP_3                          3       448         3       448          3
  BGP_Group_C3                   4       816         4       816          4
  BGP_2                          3       448         3       448          3
  BGP_Group_C2                   4       816         4       816          4
  BGP_1                          3       448         3       448          3
  BGP_Group_C1                   4       816         4       816          4
  BGP_100_100                    3       448         3       448          3
  BGP_100_100                    3       448         3       448          3
  BGP_100_100                    3       448         3       448          3
  BGP_100_100                    3       448         3       448          3
  BGP_Group_evpn                 4       816         5       828         94
  BGP_RT_Background             29    278000        32    278136        303
  bgp-thrio                      1      8192         1      8192          1
  bgp-thrio-ctx                 41     19552        60     27420     575053
  BGP addpath task               2      3584         2      3584          2
  -------------------------------------------------------------------------
              Total bytes in use:   62725488 (0% of available memory)
```

Monitoring Client EBGP Sessions

Individual route server client EBGP sessions can be viewed either as a summarized list, or specifically, using `show bgp summary` on the sample topology:

```
root@rs1> show bgp summary
Threading mode: BGP I/O
Groups: 4 Peers: 7 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.evpn.0
                      12         12          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
192.0.2.1                1       6195       6093       0       1 1d 22:26:37 Establ
  C1.inet.0: 31/31/31/0
192.0.2.2                2        332        325       0       7     2:27:38 Establ
  C2.inet.0: 31/31/31/0
192.0.2.3                3      53788      52946       0       0 2w2d 19:23:41 Establ
  C3.inet.0: 31/31/31/0

root@rs1> show bgp neighbor 192.0.2.1
Peer: 192.0.2.1+56947 AS 1    Local: 192.0.2.254+179 AS 123
  Group: C1                    Routing-Instance: C1
  Forwarding routing-instance: master
  Type: External    State: Established    Flags: <Sync>
  Last State: OpenConfirm   Last Event: RecvKeepAlive
  Last Error: Hold Timer Expired Error
  Options: <Preference AddressFamily PeerAS Refresh>
  Options: <MtuDiscovery>
  Options: <RouteServerClient>
  Address families configured: inet-unicast
  Holdtime: 90 Preference: 170
  Number of flaps: 1
  Last flap event: HoldTime
  Error: 'Hold Timer Expired Error' Sent: 1 Recv: 0
  Peer ID: 192.0.2.1        Local ID: 192.0.2.254        Active Holdtime: 90
[...Output truncated...]`
```

Monitoring Route Distribution

The following show command views the total routes present in the route server client C3's RIB, along with the RIBs where they are imported:

```
root@rs1> show route export C3.inet.0 detail
C3.inet.0                         Routes:      31
  Import: [ C1.inet.0 C2.inet.0 C3.inet.0 ]
```

To see specific routes in C3's RIB that will be exported to C1, based on the IXP global policy, view the source RIB and filter by using the target community:

```
root@rs1> show route protocol bgp community 64498:1 table C3.inet.0

C3.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

198.51.100.0/24     *[BGP/170] 2w0d 20:13:27, localpref 100
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
203.0.113.0/24      *[BGP/170] 2w0d 20:13:27, localpref 100
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
 [...Output truncated...]
```

A slightly different view, or rather a validation of the previous command, is to look at the RIB contents of all client RIBs from the perspective of what routes have been received from a specific route server client. In the next example, 192.0.2.3 is the BGP peer associated with the routing-instance C3:

```
root@rs1> show route receive-protocol bgp 192.0.2.3 | except inet.0

inet.3: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)

C1.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
  Prefix       Nexthop         MED     Lclpref    AS path
* 198.51.100.0/24         192.0.2.3                              3 I
* 203.0.113.0/24          192.0.2.3                              3 I
 [...Output truncated...]

C3.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
  Prefix       Nexthop         MED     Lclpref    AS path
* 198.51.100.0/24         192.0.2.3                              3 I
* 203.0.113.0/24          192.0.2.3                              3 I
 [...Output truncated...]
```

Specific prefixes can also be searched for to aid in troubleshooting:

```
regress@RS1> show route receive-protocol bgp 192.0.2.3 198.51.100.0 |except inet.0

C1.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
  Prefix       Nexthop         MED     Lclpref    AS path
* 198.51.100.0/24         192.0.2.3                              3 I

C3.inet.0: 31 destinations, 31 routes (31 active, 0 holddown, 0 hidden)
  Prefix       Nexthop         MED     Lclpref    AS path
* 198.51.100.0/24         192.0.2.3                              3 I
```

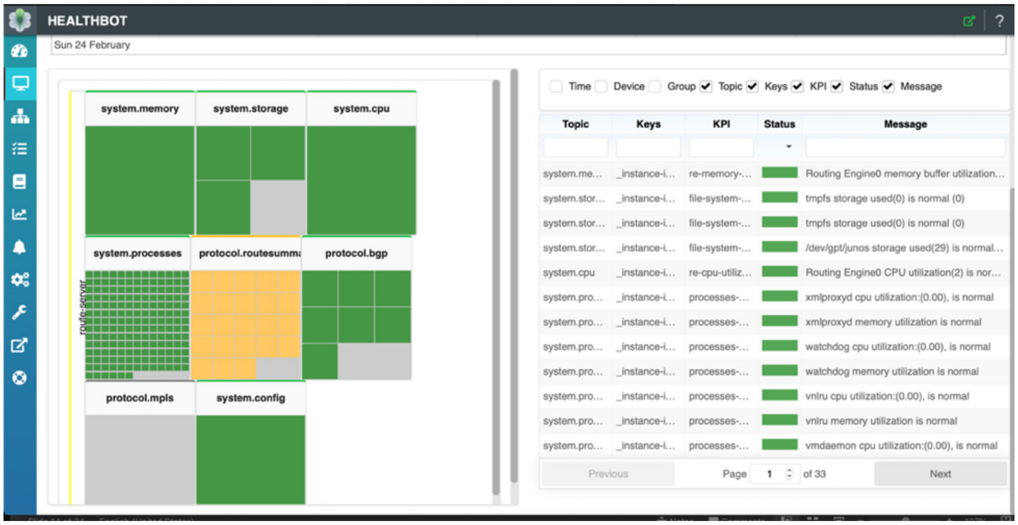
Routes may also be searched by community value or name. The search results in retrieving *all* the clients' RIBs that have a match, so route propagation between client RIBs can be tracked:

```
regress@RS1> show route protocol bgp community-name as64498_comms 198.51.100.0

inet.0: 41 destinations, 41 routes (40 active, 0 holddown, 1 hidden)

C1.inet.0: 93 destinations, 93 routes (93 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

198.51.100.0/24     *[BGP/170] 2w2d 19:32:26, localpref 100
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0

C3.inet.0: 62 destinations, 62 routes (62 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

198.51.100.0/24     *[BGP/170] 2w2d 19:32:26, localpref 100
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
```

Monitoring Tools: HealthBot

HealthBot is a highly automated and programmable device-level diagnostics and network analytics tool that provides consistent and coherent operational intelligence across network deployments. Integrated with multiple data collection methods (such as Junos Telemetry Interface, NETCONF, and SNMP), HealthBot aggregates and correlates large volumes of time-sensitive telemetry data, providing a multidimensional and predictive view of the network. Additionally, HealthBot translates troubleshooting, maintenance, and real-time analytics into an intuitive user experience to give network operators actionable insights into the health of an individual device and the overall network.

HealthBot BGP KPIs, located at https://github.com/Juniper/healthbot-rules/tree/master/juniper_official/Protocols/Bgp, contain readily consumable HealthBot playbooks and rules, which are specific to BGP neighbor key performance indicators (KPIs).



☐ Time ☐ Device ☐ Group ✓ Topic ✓ Keys ✓ KPI ✓ Status ✓ Message

Topic	Keys	KPI	Status	Message
system.me...	_instance-i...	re-memory-...	🟩	Routing Engine0 memory buffer utilization...
system.stor...	_instance-i...	file-system-...	🟩	tmpfs storage used(0) is normal (0)
system.stor...	_instance-i...	file-system-...	🟩	tmpfs storage used(0) is normal (0)
system.stor...	_instance-i...	file-system-...	🟩	/dev/gpt/junos storage used(29) is normal...
system.cpu	_instance-i...	re-cpu-utiliz...	🟩	Routing Engine0 CPU utilization(2) is nor...
system.pro...	_instance-i...	processes-...	🟩	xmlproxyd cpu utilization:(0.00), is normal
system.pro...	_instance-i...	processes-...	🟩	xmlproxyd memory utilization is normal
system.pro...	_instance-i...	processes-...	🟩	watchdog cpu utilization:(0.00), is normal
system.pro...	_instance-i...	processes-...	🟩	watchdog memory utilization is normal
system.pro...	_instance-i...	processes-...	🟩	vnlru cpu utilization:(0.00), is normal
system.pro...	_instance-i...	processes-...	🟩	vnlru memory utilization is normal
system.pro...	_instance-i...	processes-...	🟩	vmdaemon cpu utilization:(0.00), is normal

Previous Page 1 of 33 Next

Figure 3.1 HealthBot Dashboard for Route Server System KPI Monitoring

BGP KPI rules collect the statistics from network devices then analyzes the data and acts. A BGP KPI playbook is set of rules, each rule is defined with a set of KPIs. Playbooks contain BGP session state, neighbor flap detection, received routes with static threshold, and received routes with dynamic threshold rules. Rules are defined with default variable values that can be changed when deploying playbooks.

HealthBot RIB KPIs, located at https://github.com/Juniper/healthbot-rules/tree/master/juniper_official/Protocols/Rib, contain readily consumable HealthBot playbooks and rules that are specific to RIB route summary KPIs. RIB route summary KPI rules collect the statistics from network devices then analyze the data and act appropriately. The RIB route summary KPI playbook is set of rules, each rule is defined with set of KPIs. Playbooks contain route table summaries for ascertaining routes and protocol route summary rules with dynamic thresholds. Rules are defined with default variable values that can be changed while deploying the playbook.

HealthBot Systems KPIs, located at https://github.com/Juniper/healthbot-rules/tree/master/juniper_official/System, contain readily consumable HealthBot playbooks and rules that are specific to system KPIs. System KPI rules collect the statistics from network devices, then analyze the data and act. The system KPI playbook is a set of rules, where each rule is defined with a set of KPIs. Playbooks contain routing engine CPU, routing engine memory, Junos processes CPU, memory leak detection, and system storage rules. Rules are defined with default variable values, which can be changed while deploying the playbook.

Chapter 4

Using a cRPD-based Route Server

This chapter covers some special configuration considerations for using a cRPD instance or instances as a route server.

For the example below we will use the following attributes:

- Docker bridge IP subnet is 198.51.100.0/24
- IXP LAN subnet is 192.0.2.0/24
- The router-server has been assigned the IP address of 192.0.2.254 on the IXP LAN
- IXP BGP AS is 100

Special considerations for this deployment example:

- Since the EBGP clients are on a different IP subnet than the IXP LAN, the EBGP sessions *must* be multi-hop. This is due to the fact that the container runs on the Docker bridge subnet.
- You don't configure interfaces when using cRPD. The interface address is read from the Docker container.

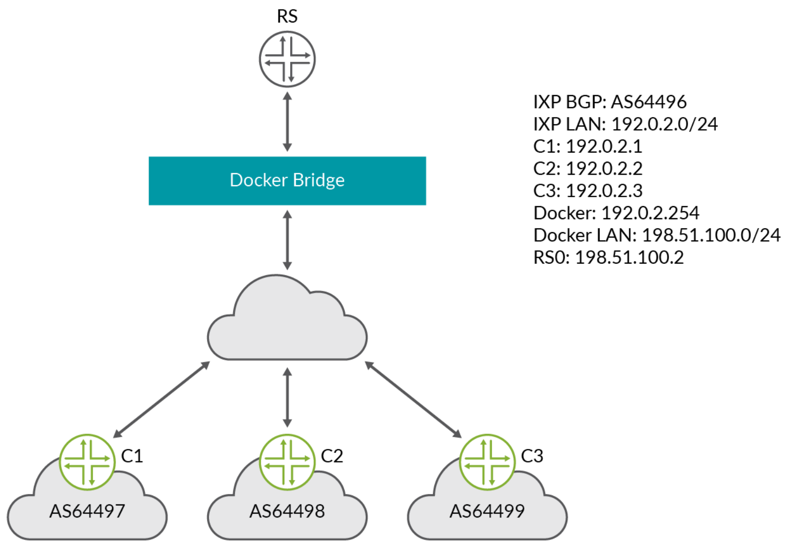


Figure 4.1 *Example IXP LAN for cRPD-based Route-Server Deployment*

Example of route-server local address:

```
root@rs0> show interfaces routing
Interface         State Addresses
eth0.0            Up    MPLS  enabled
                        ISO   enabled
                        INET  198.51.100.2
lo.0              Up    MPLS  enabled
                        ISO   enabled
                        INET  127.0.0.1
```

Example of cRPD route-server configuration:

```
root@rs1# run show configuration
routing-options {
    static {
        route 192.0.2.0/24 next-hop 198.51.100.1;
    }
    router-id 198.51.100.2;
    autonomous-system 64496;
}
protocols {
    bgp
        family inet {
            unicast {
                nexthop-resolution {
                    no-resolution;
                }
                no-install;
            }
        }
        family inet6 {
            unicast {
                nexthop-resolution {
                    no-resolution;
                }
                no-install;
            }
        }
        group IXP-members {
            type external;
            multihop;
            route-server-client;
            local-address 198.51.100.2;
            neighbor 192.0.2.1 {
                peer-as 64497;
            }
            neighbor 192.0.2.2 {
                peer-as 64498;
            }
            neighbor 192.0.2.3 {
                peer-as 64499;
```

IXP client-router configuration:

```
root@r1# run show configuration routing-options
routing-options {
    router-id 192.0.2.1;
    autonomous-system 64497;

root@r1> show configuration protocols bgp
group IXP-RS {
    type external;
    multihop;
    export export-bgp;
    neighbor 192.0.2.254 {
        local-address 192.0.2.1;
        peer-as 64496;
```

It should be noted that this example doesn't cover everything that should be configured on a client router peering with an operational route server in a live network. A good example of what should be configured on a client device running Junos OS taking part in IXP operations can be found here: https://www.ams-ix.net/ams/documentation/config-guide.

Synchronization of the Data and Control Plane

A challenging problem with IXP LANs is that the route servers are not active in the data plane between IXP member clients, so it's possible for the data plane between clients to be DOWN while the EBGP sessions with the route server remain UP. As you can imagine, this would create a blackhole – Layer 3 thinks the destination is reachable and so the client is sending traffic while Layer 2 "circuit" is down.

Work is being done in the IETF to solve this problem with BFD (https://datatracker.ietf.org/doc/draft-ietf-idr-rs-bfd/).

The abstract of the aforementioned draft states:

When BGP route servers are used, the data plane is not congruent with the control plane. Therefore, peers at an Internet exchange can lose data connectivity without the control plane being aware of it, and packets are lost. This document proposes the use of a newly defined BGP Subsequent Address Family Identifier (SAFI) both to allow the route server to request its clients use BFD to track data plane connectivity to their peers' addresses, and for the clients to signal that connectivity state back to the route server.

As of the writing of this book, there are no generally available or interoperable implementations of this draft. An external application such as HealthBot can discover and mitigate this problem as an alternative to the native BGP extensions proposed in the IETF draft.

The HealthBot Solution

The HealthBot solution works first by correlating multiple pieces of data to detect the condition. HealthBot collects and/or receives the following data:

- EBGP next hops from the route server
- EVPN MAC routes from a route collector
- Data plane OAM statistics between the PE routers

To correct the event, which is not straightforward, HealthBot does the following as illustrated in Figure 4.2:

- Restricts route distribution *only* between route server clients that are impacted by modifying instance import policies accordingly.

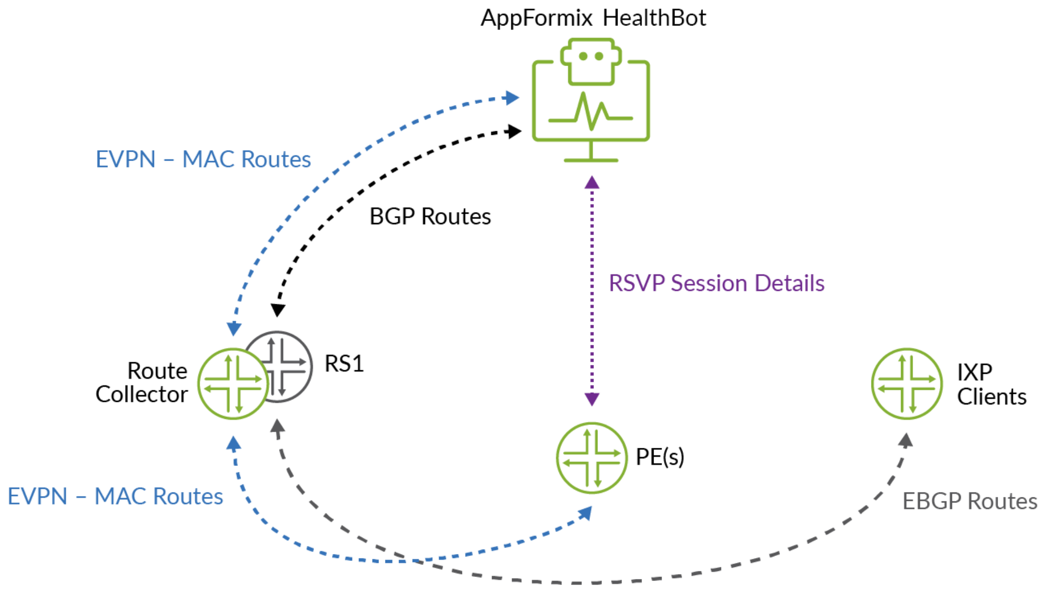


Figure 4.2 HealthBot Workflow for Validating Control Plane and Data Plane

Now let's explore a working example; consider the sample topology in Figure 4.3.

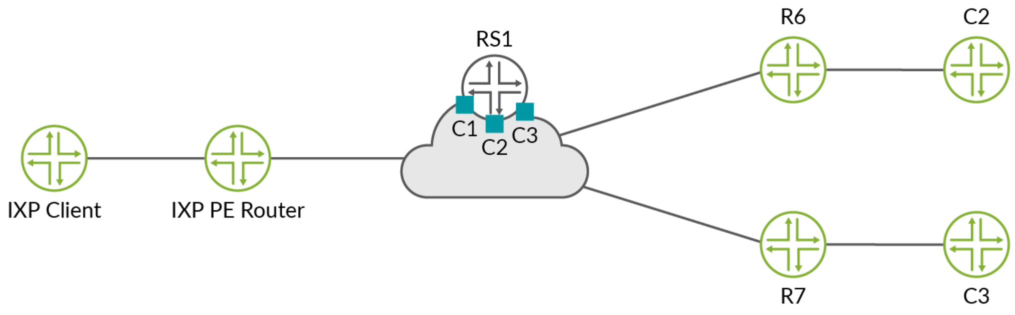


Figure 4.3 Example IXP LAN

First HealthBot subscribes to the BGP telemetry sensor and collects data from the route server (vRR) and route collector. In this example, the route server is also acting as the route collector. Additionally, HealthBot subscribes to the MPLS iAgent sensor and collect label-switched path (LSP) statistics from the PE routers R1, R6, and R7.

Finding the routing-instance names for each routing server client CLI example:

```
regress@RS1> show route instance summary | match C
Instance             Type
         Primary RIB                                 Active/holddown/hidden
C1                   non-forwarding
         C1.inet.0                                  93/0/0
C2                   non-forwarding
         C2.inet.0                                  62/0/0
C3                   non-forwarding
         C3.inet.0                                  62/0/0
```

Finding the IXP client routers IP addresses (the BGP next hops) example:

```
regress@RS1> show route receive-protocol bgp 192.0.2.6 table bgp.
evpn.0 | match “192.0.2.1|192.0.2.2|192.0.2.3”
  2:1:1::100::56:68:a6:6b:4e:e1::192.0.2.2/304 MAC/IP

regress@RS1> show route receive-protocol bgp 192.0.2.7 table bgp.
evpn.0 | match “192.0.2.1|192.0.2.2|192.0.2.3”
  2:1:1::100::56:68:a6:6b:4e:d9::192.0.2.3/304 MAC/IP

regress@RS1> show route receive-protocol bgp 192.0.2.1 table bgp.
evpn.0 | match “192.0.2.1|192.0.2.2|192.0.2.3”
  2:1:1::100::56:68:a6:6b:4e:e9::192.0.2.1/304 MAC/IP
```

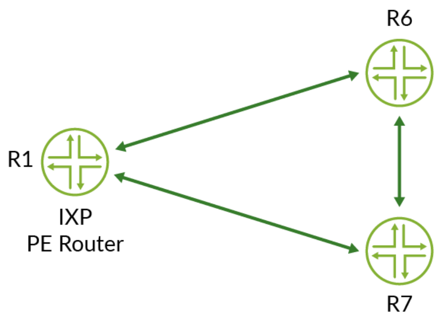


Figure 4.4 RSVP-TE LSPs Between IXP PE Routers

Referring to Figure 4.4 to ensure the data plane is intact between PE routers, HealthBot monitors the status of RSVP-TE LSPs between all IXP PE routers:

```
regress@R6> show mpls lsp | match Up
Ingress LSP: 2 sessions
To              From            State RtP     ActivePath      LSPname
192.0.2.1         192.0.2.6         Up     0 *                      to-r1
192.0.2.2         192.0.2.6         Up     0 *                      to-r2

regress@R7> show mpls lsp | match Up
Ingress LSP: 2 sessions
To              From            State RtP     ActivePath      LSPname
192.0.2.1         192.0.2.7         Up     0 *                      to-r1
192.0.2.6         192.0.2.7         Up     0 *                      to-r6

regress@R1> show mpls lsp | match Up
Ingress LSP: 2 sessions
To              From            State RtP     ActivePath      LSPname
```

```
192.0.2.6          192.0.2.1          Up      0 *                        to-r6
192.0.2.7          192.0.2.1          Up      0 *                        to-r7
```

During normal operation, routes are exchanged between IXP members according to community-based output policies. Here we can see that C1 is receiving routes from both C2 (198.51.100.0/24), and C3 (203.0.113.0/24), respectively:

```
regress@C1> **show route protocol bgp 198.51.100.1**

inet.0: 96 destinations, 96 routes (95 active, 0 holdown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

198.51.100.0/24      *[BGP/170] 2d 23:20:23, localpref100, from 192.0.2.254
                      AS path: 2 I, validation-state: unverified
                    >  to 192.0.2.2 via ge-0/0/1.0

regress@C1> **show route protocol bgp 203.0.113.0**

inet.0: 96 destinations, 96 routes (95 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

203.0.113.0/24       *[BGP/170] 3d 00:46:29, localpref100, from 192.0.2.254
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
```

However, what if one of the RSVP-TE LSPs goes DOWN, for some reason, resulting in a broken data plane between R1 and R6, but the route server is still receiving routes from IXP members attached to those PE routers?

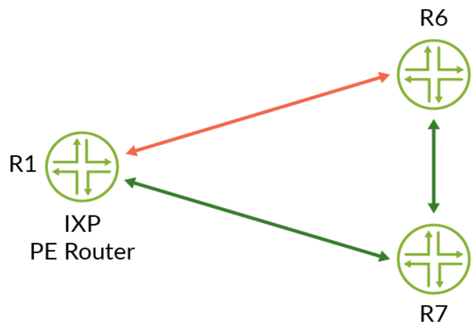


Figure 4.5 DOWN LSP Between R1 and R6

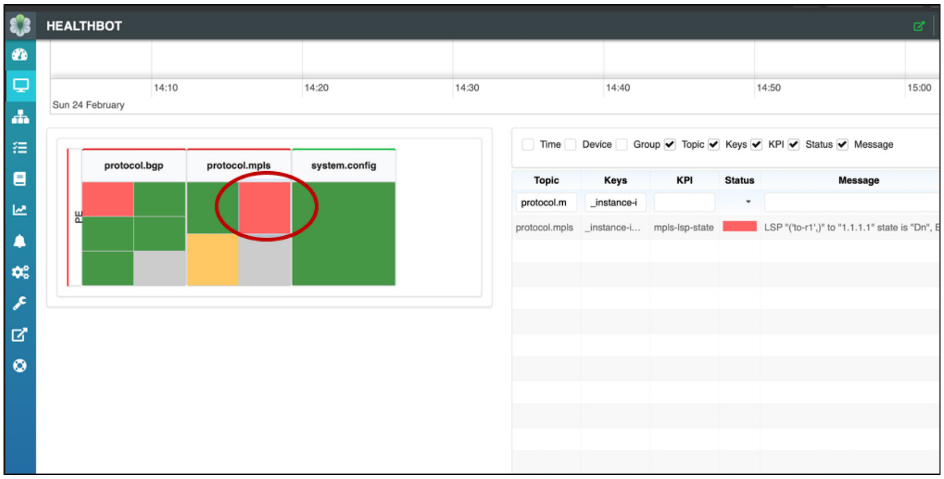
Down LSP ...

```
regress@R6# **run show mpls lsp**
Ingress LSP: 3 sessions
To                From               State RtP      ActivePath       LSPname
**192.0.2.1          0.0.0.0            Dn      0       -                to-r1**
192.0.2.7          192.0.2.6          Up      0 *                      to-r7
Total 3 displayed, Up 1, Down 2
```

But ...

```
regress@C1> show route protocol bgp 198.51.100.1

inet.0: 96 destinations, 96 routes (95 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

198.51.100.0/24     *[BGP/170] 2d 23:20:23, localpref100, from 192.0.2.254
                      AS path: 2 I, validation-state: unverified
                    >  to 192.0.2.2 via ge-0/0/1.0

regress@C1> show route protocol bgp 203.0.113.0

inet.0: 96 destinations, 96 routes (95 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

203.0.113.0/24      *[BGP/170] 3d 00:46:29, localpref100, from 192.0.2.254
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
```

HealthBot will discover the situation, display a dashboard alarm, and modify the route server policy configuration to limit route distribution between impacted IXP member routers.



☐ Time ☐ Device ☐ Group ✓ Topic ✓ Keys ✓ KPI ✓ Status ✓ Message

Topic	Keys	KPI	Status	Message
protocol.m	_instance-i			
protocol.mpls	_instance-i...	mpls-lsp-state	🟥	LSP "('to-r1',)" to "1.1.1.1" state is "Dn", E

Figure 4.6 HealthBot Dashboard with Alarm

Modified route-server configuration:

```
regress@RS1> ...ing-instances C1 routing-options instance-import
instance-import [ reject-routes-from-instance-C2 as1_comms IXP-global-policy ];

regress@RS1> ...ion routing-instances C3 routing-options instance-import
instance-import [ as3_comms IXP-global-policy ];

regress@RS1> ...ion routing-instances C2 routing-options instance-import
```

```
instance-import [reject-routes-from-instance-C1 as2_comms IXP-global-policy ];
policy-statement reject-routes-from-instance-C1 {
    term 0 {
        from instance-list C1;
        then reject;
    }
    term 1 {
        then next policy;
    }
}
policy-statement reject-routes-from-instance-C2 {
    term 0 {
        from instance-list C2;
        then reject;
    }
    term 1 {
        then next policy;
```

Verification on IXP member router C1. As you can see, routes are no longer received from member C2:

```
regress@C1> show route protocol bgp 198.51.100.1

regress@C1> show route protocol bgp 203.0.113.0

inet.0: 68 destinations, 68 routes (67 active, 0 holddown, 1 hidden)
+ = Active Route, - = Last Active, * = Both

203.0.113.0/24      *[BGP/170] 3d 01:10:07, localpref100, from 192.0.2.254
                      AS path: 3 I, validation-state: unverified
                    >  to 192.0.2.3 via ge-0/0/1.0
```

Once the data plane is restored, HealthBot will clear the alarm and revert the policy configuration.

A Dynamic and Automated Maximum Accepted Prefix Solution

As was previously discussed in the security considerations section in Chapter 1, determining, monitoring, and maintaining per route-server maximum-prefixes can be somewhat difficult for an IXP's route-server deployment. Several solutions were presented, including various multiplication factors for determining the value. HealthBot offers another solution that involves ingesting real-time streaming telemetry from the route server, maintaining per route-server client accepted prefix counts, and dynamically modifying the route-server policy when vales change. This workflow for two route-server clients is depicted in Figure 4.7.

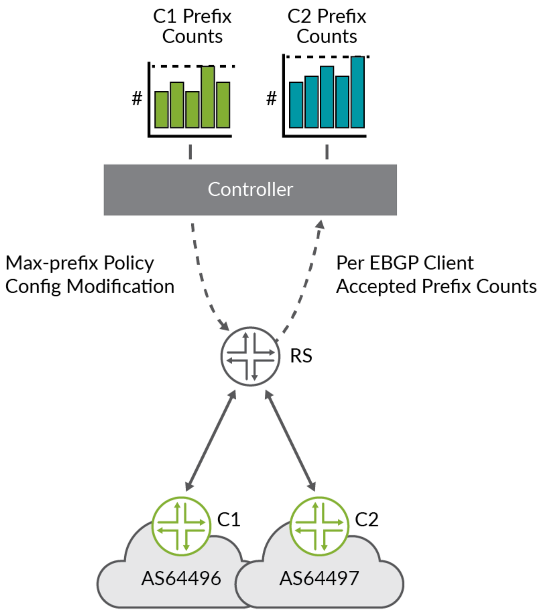
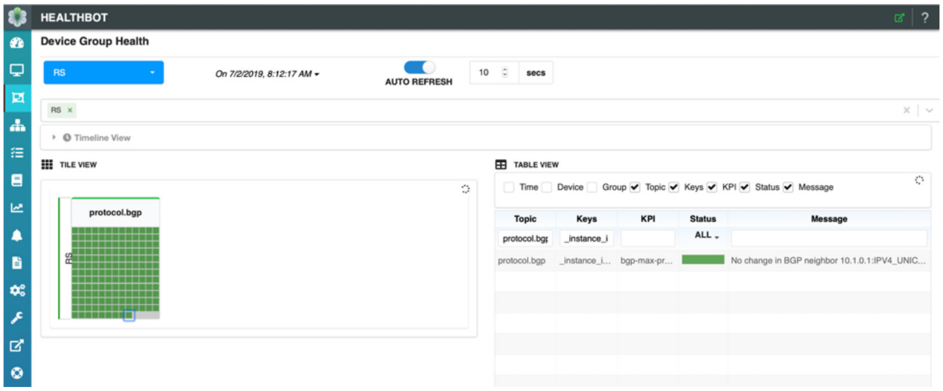

C1 Prefix Counts
C2 Prefix Counts
#
#
Controller
Max-prefix Policy Config Modification
Per EBGP Client Accepted Prefix Counts
RS
C1
C2
AS64496
AS64497

Figure 4.7 Route Server With Controller Based Dynamic Maximum Prefix Computation

Let's look at a real example. In the CLI output below you can see route server client C1. The route server is receiving and accepting 10 prefixes from client C1. You can also see that the route server is configured with a policy to accept a maximum of 15 prefixes from client C1:

```
root@rs> show bgp summary | match "Peer |C1.inet.0"
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
  C1.inet.0: 10/10/10/0

regress@rs> show configuration routing-instances C1 protocols bgp group C1 family inet
unicast {
    prefix-limit {
        maximum 15;
    }
    accepted-prefix-limit {
        maximum 15;
```

Also note the HealthBot dashboard; client C1 is selected in Figure 4.8 below and we can see that for the last statistics interval, there have been no changes to the maximum-prefix policy determined by HealthBot. This is indicated by both the 'green' status and the message in the table for that client. What is a statistics interval, you might ask? In this HealthBot solution we apply a very simple, user-defined interval over which to collect the number of accepted prefixes per route-server client. After each statistics interval, HealthBot takes the current value of accepted prefixes, multiplies by it by 1.5, also user configurable, and updates the route-server client policy with the new value.



Time Device Group [✓ Topic] [✓ Keys] [✓ KPI] [✓ Status] [✓ Message]

Topic	Keys	KPI	Status	Message
protocol.bgp	_instance_i		ALL	
protocol.bgp	_instance_i...	bgp-max-pr...	🟩	No change in BGP neighbor 10.1.0.1:IPV4_UNIC...

Figure 4.8 Healthbot Per-Client Maximum Prefix Values

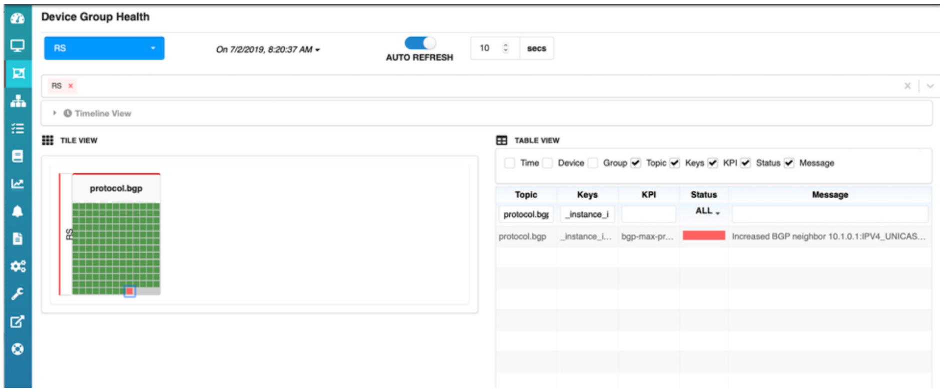
Now, let's have client C1 advertise a few more routes to the route server and add two more static routes to C1's BGP export policy:

```
root@c1> set routing-options static route 198.51.100.1/32 next-hop 192.0.2.1
root@c1> set routing-options static route 198.51.100.2/32 next-hop 192.0.2.1

root@r1> show configuration logical-systems R1 policy-options policy-statement ebgp-out
term announce {
    from {
        route-filter 192.0.2.0/24 orlonger;
    }
    then {
        community add out-comms;
        accept;
    }
}
then reject;
```

We can see that the additional prefixes are received and accepted by the route server in the output below:

```
root@rs> show bgp summary | match "Peer |C1.inet.0"
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
r1001.inet.0: 12/12/12/0
```

But look at the HealthBot dashboard now. The icon for client C1 has turned red. This is because the number of accepted prefixes has exceeded the threshold of 90%, also user configurable.



Device Group Health

RS — On 7/2/2019, 8:20:37 AM — AUTO REFRESH 10 secs

RS ×

Timeline View

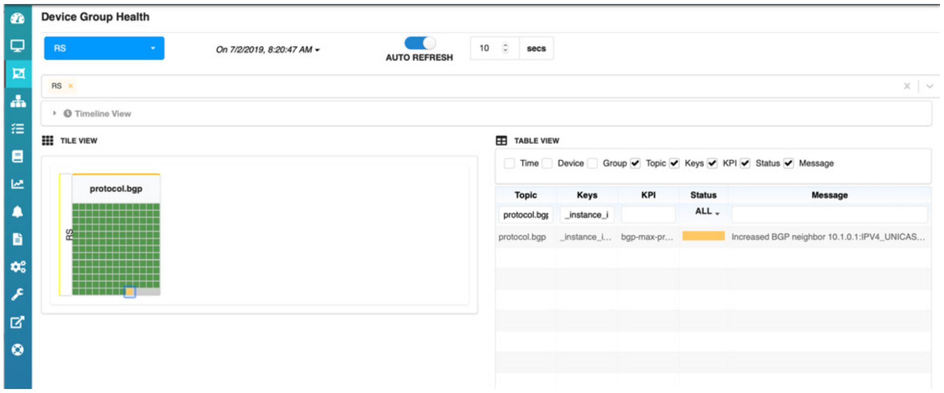
TILE VIEW: protocol.bgp, RS

TABLE VIEW: Time, Device, Group ✓, Topic ✓, Keys ✓, KPI ✓, Status ✓, Message

Topic	Keys	KPI	Status	Message
protocol.bgp	_instance_i		ALL	
protocol.bgp	_instance_i...	bgp-max-pr...	🟥	Increased BGP neighbor 10.1.0.1:IPV4_UNICAS...

Figure 4.9 Healthbot Dahsboard Showing Maximum Prefix Validation

At the end of the statistics collection interval, we can also see that C1's icon has now turned yellow, this indicates that HealthBot is changing the maximum-prefix policy for client C1 on the route server. And the HealthBot dashboard will again show C1's icon as green since it is within policy and not exceeding any thresholds.



Device Group Health

RS — On 7/2/2019, 8:20:47 AM — AUTO REFRESH 10 secs

RS ×

Timeline View

TILE VIEW: protocol.bgp, RS

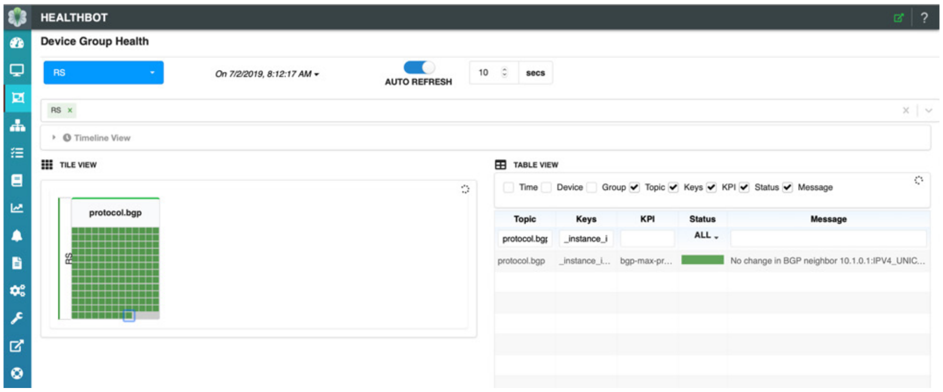
TABLE VIEW: Time, Device, Group ✓, Topic ✓, Keys ✓, KPI ✓, Status ✓, Message

Topic	Keys	KPI	Status	Message
protocol.bgp	_instance_i		ALL	
protocol.bgp	_instance_i...	bgp-max-pr...	🟨	Increased BGP neighbor 10.1.0.1:IPV4_UNICAS...

Figure 4.10 C1's Icon is Green

As you can see, the route-server configuration has been updated with the new value (12 accepted prefix * 1.5) of 18:

```
root@rs> show configuration routing-instances C1 protocols bgp group C1 family inet
unicast {
    prefix-limit {
        maximum 18;
    }
    accepted-prefix-limit {
        maximum 21;
```



☐ Time ☐ Device ☐ Group ☑ Topic ☑ Keys ☑ KPI ☑ Status ☑ Message

Topic	Keys	KPI	Status	Message
protocol.bgp	_instance_i		ALL	
protocol.bgp	_instance_i...	bgp-max-pr...	🟩	No change in BGP neighbor 10.1.0.1:IPV4_UNIC...

Figure 4.11 *HealthBot Dashboard Showing Number of Accepted Prefix*

Lastly, each route-server client's accepted prefixes and corresponding max-prefix value can be individually monitored. In Figure 4.12, you can see the number of accepted prefixes for client C1 along with the max-prefix policy value configured during the statistics interval.

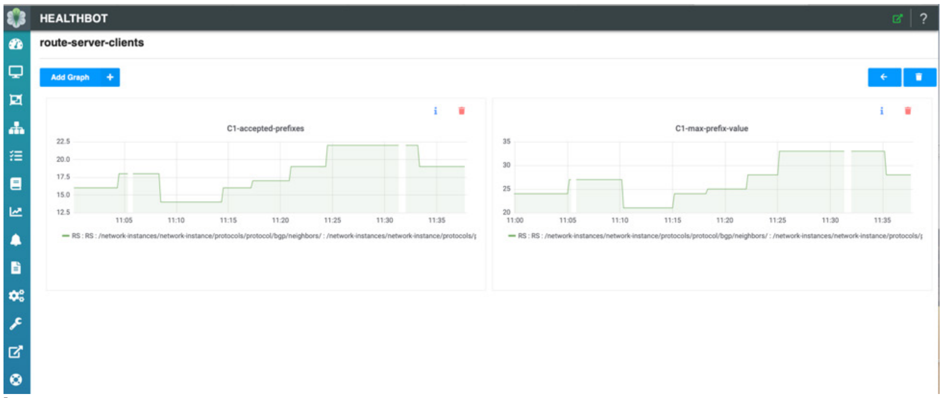

HEALTHBOT
route-server-clients
Add Graph
C1-accepted-prefixes
22.5
20.0
17.5
15.0
12.5
11:05
11:10
11:15
11:20
11:25
11:30
11:35
RS : RS : /network-instances/network-instance/protocols/protocol/bgp/neighbors/ : /network-instances/network-instance/protocols/
C1-max-prefix-value
35
30
25
20
11:00
11:05
11:10
11:15
11:20
11:25
11:30
11:35
RS : RS : /network-instances/network-instance/protocols/protocol/bgp/neighbors/ : /network-instances/network-instance/protocols/

Figure 4.12 *HealtBot Dashbaord Showing History of Accepted and Computed Maximum Prefix Values*

Summary

The authors hope that the information provided in this book will help you get started designing, testing, and deploying Junos OS-based route servers. The configuration and design considerations discussed in this book are simply to get you started. In the end every IXP is unique and will have its own considerations for implementing policy.

Appendixes

Appendix A: Mutually Agreed Upon Norms for Routing Security (MANRS)

By now you should have an idea how to deploy a Junos OS-based route server. Or maybe you have even built one already?

If so, did you take into account the MANRS? MANRS is a global initiative, supported by the Internet Society, that provides crucial fixes to reduce the most common routing threats. There's a special section for IXPs on their website: https://www.manrs.org/ixps/#actions.

For example, MANRS asks you to prevent propagation of incorrect routing information or offer assistance to the IXP members to maintain accurate routing information in an appropriate repository (IRR and/or RPKI).

Joining MANRS, actively implementing the guidelines and promoting it shows that you care about the stability and security of the Internet.

Appendix B: Relevant Standardization Documents

Most of what we've described in this book is to be found in Requests for Comments (RFCs). We're linking some of them here for reference and further reading.

- RFC7947: Internet Exchange BGP Route Server
  https://tools.ietf.org/html/rfc7947
- RFC7948: Internet Exchange BGP Route Server Operations
  https://tools.ietf.org/html/rfc7948
- RFC1997: BGP Community Attribute
  https://tools.ietf.org/html/rfc1997
- RFC4360: BGP Extended Communities Attribute
  https://tools.ietf.org/html/rfc4360
- RFC3682: Generalized TTL Security Mechanism
  https://tools.ietf.org/html/rfc3682
- RFC8212: Default eBGP Route Propagation Behavior Without Policies
  https://tools.ietf.org/html/rfc8212
- BGP Maximum Prefix Limits Draft
  https://tools.ietf.org/html/draft-sa-grow-maxprefix-02

Appendix C: Emerging Technologies

Ethernet VPNs

Ethernet VPN (EVPN) enables you to connect dispersed customer sites using a Layer 2 virtual bridge. EVPN operates in contrast to the existing virtual private LAN service (VPLS) by enabling control plane-based MAC learning in the core. In EVPN, PEs participating in the EVPN instances learn customer MAC routes in the control plane using MP-BGP protocol.

Control plane MAC learning brings a number of benefits that allow EVPN to address the VPLS shortcomings, including support for multi-homing with per-flow load balancing.

EVPN provides the following benefits:

- *Integrated Services*: Integrated Level 2 and Level 3 VPN services, L3VPN-like principles and operational experience for scalability and control, all-active multi-homing and PE load-balancing using ECMP, and load balancing of traffic to and from Customer Edge routers (CEs) that are multihomed to multiple PEs.

- *Network Efficiency*: Eliminates flood and learn mechanism, fast-reroute, resiliency, and faster reconvergence when the link to a dual-homed server fails, optimized Broadcast, Unknown-unicast, and Multicast (BUM) traffic delivery.
- *Service Flexibility*: MPLS data plane encapsulation, support existing and new services types (E-LAN, E-Line), peer PE auto-discovery, and redundancy group auto-sensing.

The following EVPN modes are supported:

- *Single-homing*: This enables you to connect a CE device to one PE device.
- *Multihoming*: This enables you to connect a CE device to more than one PE device. Multihoming ensures redundant connectivity. The redundant PE device ensures that there is no traffic disruption when there is a network failure. Following are the types of multihoming:
- *All-Active*: In all-active mode, all the PEs attached to the particular Ethernet segment is allowed to forward traffic to and from that Ethernet segment.

MORE? For more information see: https://www.juniper.net/uk/en/training/jnbooks/day-one/proof-concept-labs/using-ethernet-vpns/ as a proof of concept straight from Juniper's Proof of Concept Labs (POC Labs). The book supplies a sample topology, all the configurations, validation testing, as well as some high availability (HA) tests.

Routing in Fat Trees (RIFT)

IXPs have been steadily growing and connecting more networks within a single L2 network. Because of their topologies (traditional and emerging), IXP networks have a unique set of requirements for traffic patterns that need fast restoration and low human intervention.

Lately Clos and Fat-Tree topologies have gained popularity in data center networks as a result of a trend towards centralized data center network architectures that may deliver computation and storage services. It may be worth investigating whether such a topology could be suitable for an IXP environment as well.

The Routing in Fat Trees (RIFT) protocol addresses the demands of routing in Clos and Fat-Tree networks via a mixture of both link-state and distance-vector techniques colloquially described as *link-state towards the spine, and distance vector towards the leafs*. RIFT uses this hybrid approach to focus on networks with regular topologies with a high degree of connectivity, a defined directionality, and large scale.

The RIFT protocol:

- Provides automatic construction of fat-tree topologies based on detection of links,
- Minimizes the amount of routing state held at each topology level,
- Automatically prunes topology distribution exchanges to a sufficient subset of links,
- Support automatic disaggregation of prefixes on link and node failures to prevent black-holing and suboptimal routing,
- Allows traffic steering and re-routing policies, and
- Provides mechanisms to synchronize a limited key-value data-store that can be used after protocol convergence.

Nodes participating in the protocol need only very light configuration and should be able to join a network as leaf nodes simply by connecting to the network using default configuration.

NOTE For more information on RIFT see https://datatracker.ietf.org/doc/draft-ietf-rift-rift/ .

VXLAN

VXLAN is a MAC in IP/UDP (MAC-in-UDP) encapsulation technique with a 24-bit segment identifier in the form of a VXLAN ID. The larger VXLAN ID allows LAN segments to scale to 16 million in a cloud network. In addition, the IP/UDP encapsulation allows each LAN segment to be extended across existing Layer 3 networks making use of Layer 3 ECMP.

VXLAN provides a way to extend Layer 2 networks across Layer 3 infrastructure using MAC-in-UDP encapsulation and tunneling. VXLAN enables flexible workload placements using the Layer 2 extension. It can also be an approach to building a multi-tenant data center by decoupling tenant Layer 2 segments from the shared transport network.

VXLAN has the following benefits:

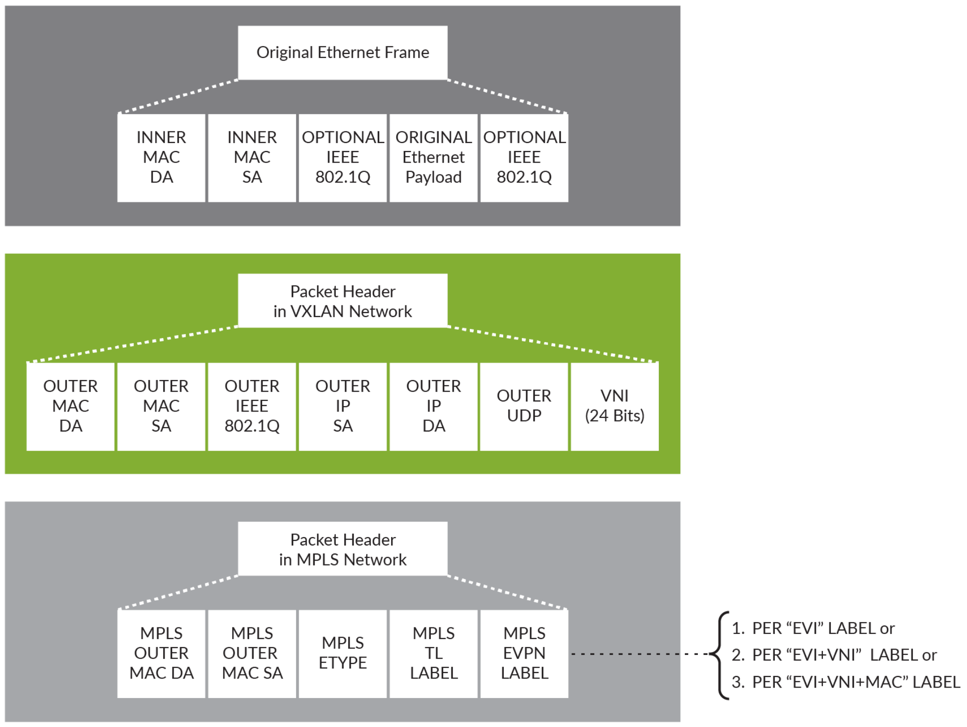
- Flexible placement of multi-tenant segments throughout the data center.
- It provides a way to extend Layer 2 segments over the underlying shared network infrastructure so that tenant workloads can be placed across physical pods in the data center.
- Higher scalability to address more Layer 2 segments.

- VXLAN uses a 24-bit segment ID, the VXLAN network identifier (VNID). This allows a maximum of 16 million VXLAN segments to coexist in the same administrative domain. (In comparison, traditional VLANs use a 12-bit segment ID that can support a maximum of 4096 VLANs.)
- Utilization of available network paths in the underlying infrastructure.

VXLAN packets are transferred through the underlying network based on its Layer 3 header. It uses ECMP routing and link aggregation protocols to use all available paths.

VXLAN is often described as an overlay technology because it allows you to stretch Layer 2 connections over an intervening Layer 3 network by encapsulating (tunneling) Ethernet frames in a VXLAN packet that includes IP addresses. Devices that support VXLANs are called virtual tunnel endpoints (VTEPs)—they can be end hosts or network switches or routers. VTEPs encapsulate VXLAN traffic and de-encapsulate that traffic when it leaves the VXLAN tunnel. To encapsulate an Ethernet frame, VTEPs add a number of fields, including the following fields:

- Outer media access control (MAC) destination address (MAC address of the tunnel endpoint VTEP)
- Outer MAC source address (MAC address of the tunnel source VTEP)
- Outer IP destination address (IP address of the tunnel endpoint VTEP)
- Outer IP source address (IP address of the tunnel source VTEP)
- Outer UDP header
- A VXLAN header that includes a 24-bit field – called the VXLAN network identifier (VNI) – that is used to uniquely identify the VXLAN. The VNI is similar to a VLAN ID, but having 24 bits allows you to create many more VXLANs than VLANs.



DA	Destination Address
EVI	EVPN Instance
SA	Source Address
TL	Tunnel Label
VNI	VXLAN Network Identifier

Figure A.1 Difference between Ethernet Frame, VXLAN Header, and, for Reference, the MPLS Header

MORE? For more information see *Understanding EVPN with VXLAN Data Plane Encapsulation*: https://www.juniper.net/documentation/en_US/junos/topics/concept/evpn-vxlan-data-plane-encapsulation.html.