

cRPD Deployment Guide for Linux Server

Published
2019-12-26

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

cRPD Deployment Guide for Linux Server
Copyright © 2019 Juniper Networks, Inc. All rights reserved.

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at <https://support.juniper.net/support/eula/>. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About the Documentation | vii

Documentation and Release Notes | vii

Documentation Conventions | vii

Documentation Feedback | x

Requesting Technical Support | x

Self-Help Online Tools and Resources | xi

Creating a Service Request with JTAC | xi

1

Overview

Understanding Containerized RPD | 15

Routing Protocol Process | 15

Routing Engine Kernel | 15

cRPD Overview | 16

Benefits | 17

Docker Overview | 18

Supported Features on cRPD | 19

Licensing | 20

Use case: Egress Peer Traffic Engineering using BGP Add-Path | 20

cRPD Resource Requirements | 21

cRPD Scaling | 22

2

Installing and Upgrading cRPD

Requirements for Deploying cRPD on a Linux Server | 25

Host Requirements | 25

Interface Naming and Mapping | 25

Installing cRPD on Docker | 26

Before You Install | 26

Install and Verify Docker | 27

Download the cRPD Software | 27

Creating Data Volumes and Running cRPD using Docker | 28

Configuring cRPD using the CLI | 29

Configuring Memory | 33

Upgrading cRPD | 33

Upgrade Software | 33

3

Managing cRPD

Managing cRPD | 37

Viewing Container Processes in a Running cRPD | 38

Accessing cRPD CLI and Bash Shell | 38

Pausing and Resuming Processes within a cRPD Container | 39

Removing a cRPD Instance | 39

Viewing Docker Statistics and Logs | 40

Viewing Active Containers | 41

Stopping the Container | 41

Establishing an SSH Connection for a NETCONF Session and cRPD | 41

Establishing an SSH Connection | 42

Enabling SSH | 42

Port Forwarding Mechanism | 43

Connecting to a NETCONF Server on Container | 43

4

Programmable Routing

cRPD Application Development Using JET APIs | 47

JET APIs | 47

Getting Started with JET | 47

Configure JET Interaction with Linux OS | 48

Maximum Number of JET Connections | 48

Compile IDL Files | 48

5

Configuring cRPD Features

Configuring Settings on Host OS | 53

Configuring ARP Scaling | 53

Configuring OSPFv2/v3 | 54

Configuring MPLS | 54

Adding MPLS Routes | 55

- Adding Routes with MPLS label | 55
- Creating a VRF device | 56
- Assigning a Network Interface to a VRF | 56
- Viewing the Devices assigned to VRF | 57
- Viewing Neighbor Entries to VRF | 57
- Viewing Addresses for a VRF | 57
- Viewing Routes for a VRF | 58
- Removing Network Interface from a VRF | 59

Multitopology Routing in cRPD | 59

- Understanding Multitopology in cRPD | 59
- Example: Configuring Multitopology Routing with BGP in cRPD | 60

Layer 3 Overlay Support in cRPD | 66

- Understanding Layer 3 Overlay VRF support in cRPD | 67
- Moving the Interfaces under a VRF | 68
- Example: Configuring Layer 3 VPN (VRF) on cRPD Instance | 68

MPLS Support in cRPD | 80

- Understanding MPLS support in cRPD | 80
- Example: Configuring Static Label Switched Paths for MPLS in cRPD | 81

6

Troubleshooting

Debugging cRPD Application | 97

- Command-Line Interface | 97
- Fault Handling | 98
- Troubleshooting Container | 98
- Verify Docker | 99
- Viewing Core Files | 100
- Configuring Syslog | 100

About the Documentation

IN THIS SECTION

- Documentation and Release Notes | vii
- Documentation Conventions | vii
- Documentation Feedback | x
- Requesting Technical Support | x

Use this guide to install the containerized routing protocol process (cRPD) in the Linux environment. This guide also includes basic cRPD container configuration and management procedures.

After completing the installation, management, and basic configuration procedures covered in this guide, refer to the Junos OS documentation for information about further software configuration.

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at <https://www.juniper.net/documentation/>.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at <https://www.juniper.net/books>.

Documentation Conventions

Table 1 on page viii defines notice icons used in this guide.

Table 1: Notice Icons







Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page viii defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the configure command: user@host> configure
Fixed-width text like this	Represents output that appears on the terminal screen.	user@host> show chassis alarms No alarms currently active
<i>Italic text like this</i>	<ul style="list-style-type: none"> Introduces or emphasizes important new terms. Identifies guide names. Identifies RFC and Internet draft titles. 	<ul style="list-style-type: none"> A policy <i>term</i> is a named structure that defines match conditions and actions. <i>Junos OS CLI User Guide</i> RFC 1997, <i>BGP Communities Attribute</i>

Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
<i>Italic text like this</i>	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# set system domain-name <i>domain-name</i>
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	<ul style="list-style-type: none"> To configure a stub area, include the stub statement at the [edit protocols ospf area area-id] hierarchy level. The console port is labeled CONSOLE.
< > (angle brackets)	Encloses optional keywords or variables.	stub <default-metric <i>metric</i> >;
(pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	broadcast multicast (<i>string1</i> <i>string2</i> <i>string3</i>)
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	rsvp { # Required for dynamic MPLS only
[] (square brackets)	Encloses a variable for which you can substitute one or more values.	community name members [<i>community-ids</i>]
Indentation and braces ({ })	Identifies a level in the configuration hierarchy.	[edit] routing-options { static { route default { nexthop <i>address</i> ; retain; } } }
;(semicolon)	Identifies a leaf statement at a configuration hierarchy level.	

GUI Conventions

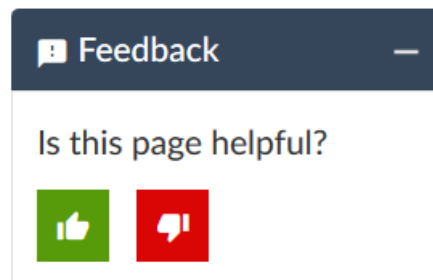
Table 2: Text and Syntax Conventions (*continued*)

Convention	Description	Examples
Bold text like this	Represents graphical user interface (GUI) items you click or select.	<ul style="list-style-type: none"> In the Logical Interfaces box, select All Interfaces. To cancel the configuration, click Cancel.
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select Protocols>Ospf .

Documentation Feedback

We encourage you to provide feedback so that we can improve our documentation. You can use either of the following methods:

- Online feedback system—Click TechLibrary Feedback, on the lower right of any page on the [Juniper Networks TechLibrary](#) site, and do one of the following:



- Click the thumbs-up icon if the information on the page was helpful to you.
- Click the thumbs-down icon if the information on the page was not helpful to you or if you have suggestions for improvement, and use the pop-up form to provide feedback.
- E-mail—Send your comments to techpubs-comments@juniper.net. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active Juniper Care or Partner Support Services support contract, or are

covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at <https://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf>.
- Product warranties—For product warranty information, visit <https://www.juniper.net/support/warranty/>.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: <https://www.juniper.net/customers/support/>
- Search for known bugs: <https://prsearch.juniper.net/>
- Find product documentation: <https://www.juniper.net/documentation/>
- Find solutions and answer questions using our Knowledge Base: <https://kb.juniper.net/>
- Download the latest versions of software and review release notes: <https://www.juniper.net/customers/csc/software/>
- Search technical bulletins for relevant hardware and software notifications: <https://kb.juniper.net/InfoCenter/>
- Join and participate in the Juniper Networks Community Forum: <https://www.juniper.net/company/communities/>
- Create a service request online: <https://myjuniper.juniper.net>

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: <https://entitlementsearch.juniper.net/entitlementsearch/>

Creating a Service Request with JTAC

You can create a service request with JTAC on the Web or by telephone.

- Visit <https://myjuniper.juniper.net>.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see <https://support.juniper.net/support/requesting-support/>.

1

CHAPTER

Overview

Understanding Containerized RPD | 15

cRPD Resource Requirements | 21

Understanding Containerized RPD

Containerized routing protocol process (cRPD) is Juniper's routing protocol process (rpd) decoupled from Junos OS and packaged as a Docker container to run in Linux-based environments. rpd runs as user space application and learns route state through various routing protocols and maintains the complete set in routing information base (RIB), also known as routing table. The rpd process is also responsible for downloading the routes into the forwarding information base (FIB) also known as forwarding table based on local selection criteria. Whereas the Packet Forwarding Engine (PFE) in Juniper Networks router holds the FIB and does packet forwarding, for cRPD. The host Linux kernel stores the FIB and performs packet forwarding. cRPD can also be deployed to provide control plane-only services such as BGP route reflection.

NOTE: Route Reflection networking service must not depend on the same hardware or the controllers that host the application software containers that are using the reachability learnt by using the Route Reflection service. cRR service must work independently.

Routing Protocol Process

Within Junos OS, the routing protocol process controls the routing protocols that run on a router. The rpd process starts all configured routing protocols and handles all routing messages. It maintains one or more routing tables, which consolidate the routing information learned from all routing protocols. From this routing information, the routing protocol process determines the active routes to network destinations and installs these routes into the Routing Engine's forwarding table. Finally, rpd implements a routing policy, which enables you to control the routing information that is transferred between the routing protocols and the routing table. Using the routing policy, you can filter and limit the transfer of information as well as set properties associated with specific routes.

Routing Engine Kernel

The Routing Engine software consists of several software processes that control router functionality and a kernel that provides the communication among all the processes.

The Routing Engine kernel provides the link between the routing tables and the Routing Engine's forwarding table. The kernel is also responsible for all communication with the Packet Forwarding Engine, which includes keeping the Packet Forwarding Engine's copy of the forwarding table synchronized with the master copy in the Routing Engine.

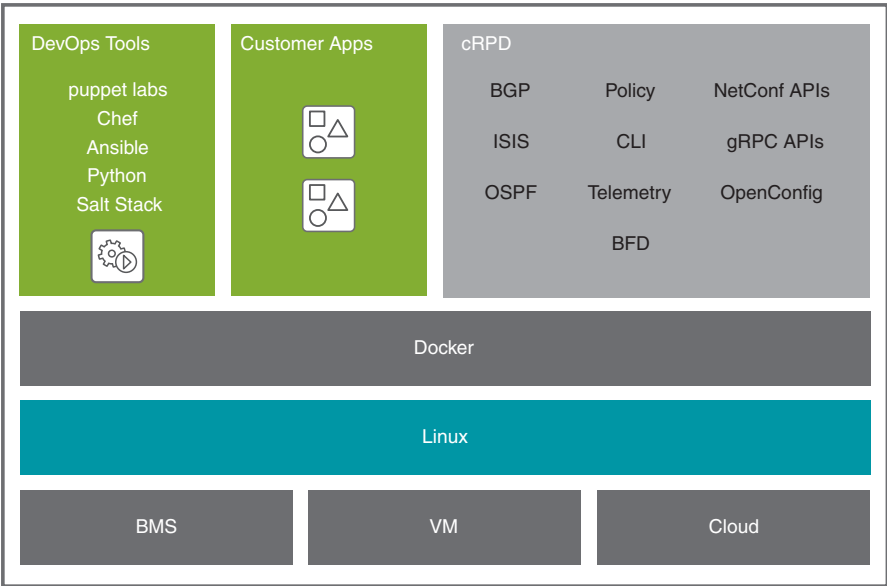
The RPD runs natively on Linux and communicates program routes into the Linux kernel using Netlink. The Netlink messages are used to install FIB state generated by RPD to Linux kernel, to interact with mgd and cli for configuration and management, to maintain protocol sessions using PPMD, and to detect liveness using BFD. RPD learns the interface attributes such as their name, addresses, MTU settings and link status from the netlink messages. Netlink acts as an interface to the kernel components.

cRPD Overview

cRPD maintains the route state information in RIB and forwards the routes into FIB based on local route selection criteria. cRPD contains the RPD, PPMD, CLI, MGD, and BFD processes. cRPD defines how routing protocols such as ISIS, OSPF, and BGP operate on the device, including selecting routes and maintaining forwarding tables.

Figure 1 on page 16 shows the cRPD overview.

Figure 1: cRPD Overview



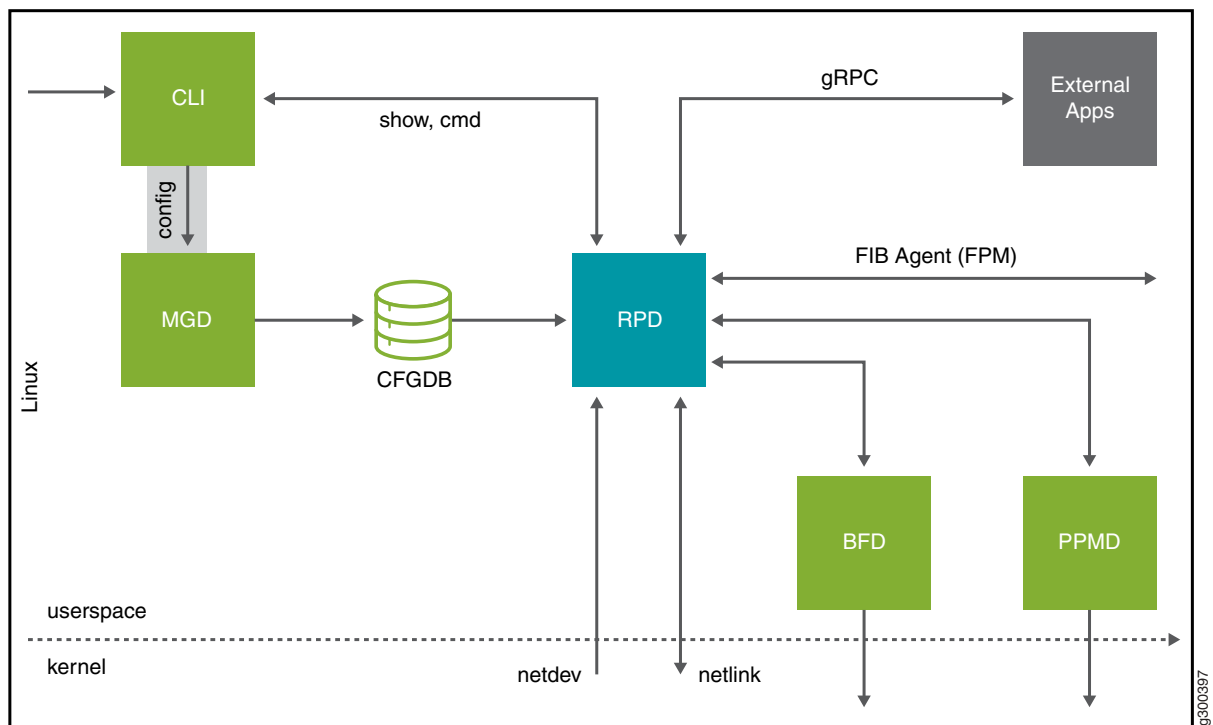
The network interfaces learnt by the underlying OS kernel are exposed to the RPD on Linux container. RPD learns about all the network interfaces and add route state for all the network interfaces. If there are additional Docker containers running in the system, all the containers and application running directly on the host can access to the same set of network interfaces and state.

When multiple cRPD running on a system, that is in bridge mode, containers are connected to the host network stack through bridges. cRPD is connected to the Linux OS using bridges. The host interfaces are connected to the container using bridges. Multiple containers can connect to the same bridge and

communicate with one another. The default Docker bridge enables NAT. NAT bridges are used as a management port into the container. This means the clients cannot initiate connections to the cRR and cRR is in control of which clients it connects to. If the bridge is connected to the host OS network interfaces, external communication is feasible. For routing purposes, it is possible to assign all or a subset of physical interfaces for use by a Docker container. This mode is useful for containerized Route Reflector and partitioning the system into multiple routing domains.

Figure 2 on page 17 shows the architecture of cRPD able to run natively on Linux.

Figure 2: cRPD on Linux Architecture



Benefits

- The use of containers reduces the time required for service boot up from several minutes to a few seconds, which results in faster deployment.
- You can run cRPD on any Linux server that supports Docker.
- With a small footprint and minimum resource reservation requirements, cRPD can easily scale to keep up with customers' peak demand.

- Provides significantly higher density without requiring resource reservation on the host than what is offered by VM-based solutions.
- Well proven or a stable routing software on Linux with cRPD.

Docker Overview

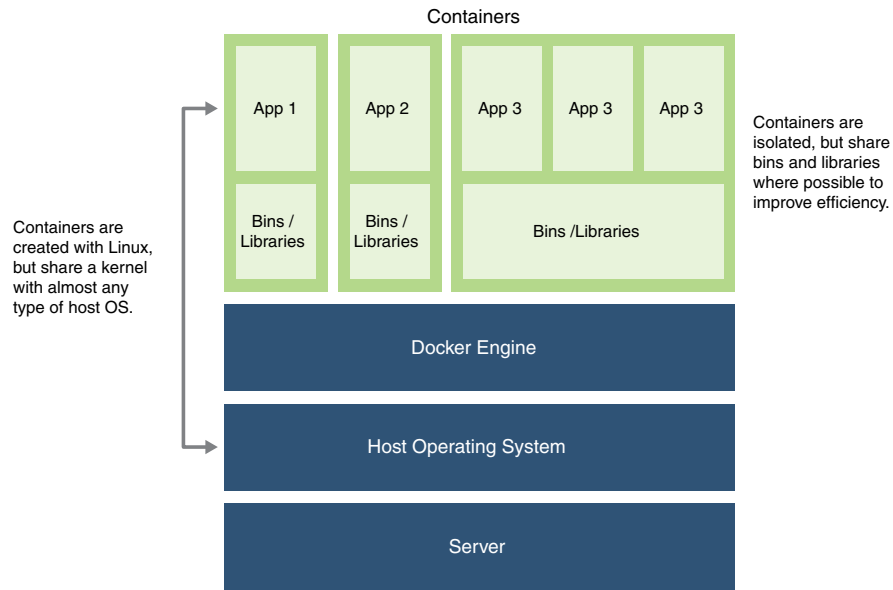
Docker is an open source software platform that simplifies the creation, management, and teardown of a virtual container that can run on any Linux server. A Docker container is an open source software development platform, with its main benefit being to package applications in “containers” to allow them to be portable among any system running the Linux operating system (OS). A container provides an OS-level virtualization approach for an application and associated dependencies that allow the application to run on a specific platform. Containers are not VMs, rather they are isolated virtual environments with dedicated CPU, memory, I/O, and networking.

A container image is a lightweight, standalone, executable package of a piece of software. Because containers include all dependencies for an application, multiple containers with conflicting dependencies can run on the same Linux distribution. Containers use the host OS Linux kernel features, such as groups and namespace isolation, to allow multiple containers to run in isolation on the same Linux host OS. An application in a container can have a small memory footprint because the container does not require a guest OS, which is required with VMs, because it shares the kernel of its Linux host's OS.

Containers have a high spin-up speed and can take much less time to boot up as compared to VMs. This enables you to install, run, and upgrade applications quickly and efficiently.

[Figure 3 on page 19](#) provides an overview of a typical Docker container environment.

Figure 3: Docker Container Environment



Supported Features on cRPD

cRPD supports the following features:

- BGP Route Reflector in the Linux container
- BGP add-path, multipath, graceful restart helper mode
- BGP, OSPF, OSPFv3, IS-IS, and Static
- BMP, BFD, and Linux-FIB
- Equal-Cost Multipath (ECMP)
- JET for Programmable RPD
- Junos CLI support
- Management using open interfaces NETCONF, and SSH
- Support for IPv4 and IPv6 routing

Licensing

The cRPD software features require a license to activate the feature. To understand more about cRPD licenses, see [Supported Features on cRPD](#), [Juniper Agile Licensing Guide](#), and [Managing cRPD Licenses](#).

Use case: Egress Peer Traffic Engineering using BGP Add-Path

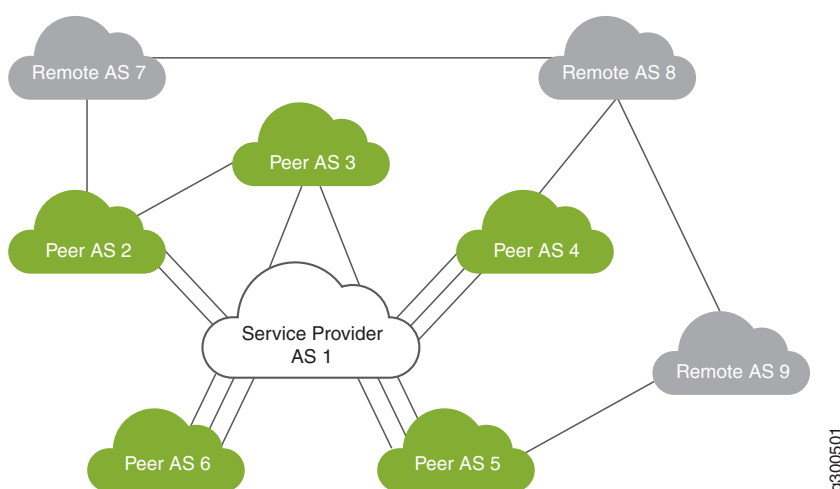
Service providers meet growing traffic demands. They need services which keep their capital expenditure and operating expenditure low. Juniper provides tools and applications to deploy, configure, manage and maintain this complexity.

Egress peer traffic engineering (TE) allows a central controller to instruct an ingress router in a domain to direct traffic towards a specific egress router and a specific external interface to reach a particular destination out of the network for optimum utilization of the advertised egress routes.

The Internet – a public global network of networks – is built as system of interconnected networks of Service Provider (SP) infrastructures. These networks are often represented as Autonomous Systems (ASs) each has globally unique Autonomous System Number (ASN). The data-plane interconnection link (NNI) and control-plane (eBGP) direct connection between two ASs allows Internet traffic to travel between the two, usually as part of a formal agreement called peering.

A SP has multiple peering relationship with multiple other SPs. They are usually geographically distributed, differ in number and bandwidth of the NNI link, and use various business or cost models

Figure 4: Peering Among Service Providers



In the context of AS peering, traffic egress assumes that the destination network address is reachable through a certain peer AS. So, for example, a device in Peer AS#2 can reach a destination IP address in Peer AS#4 through Service Provider AS#1. This reachability information is provided by a peer AS using an eBGP Network Layer Reachability Information (NLRI) advertisement. An AS typically advertises IP addresses that belong to it, but an AS may also advertise addresses learned from another AS. For example, Peer AS#2 can advertise to SP (AS#1) addresses it has received from Peer AS#3, Peer AS#7 and even Peer AS#8, Peer AS#9, Peer AS#4 and Peer AS#5. It all depends of the BGP routing policies between the individuals ASs. Therefore, a given destination IP prefix can be reached through multiple peering ASs and over multiple NNIs. It is the role of routers and network operators in the SP network to select “best” exit interface for each destination prefix.

The need for engineering the way that traffic exits the service provider AS is critical for ensuring cost efficiency while providing a good end user experience at the same time. The definition of “best” exit interface is a combination of cost as well as latency and traffic loss.

For more information, see [Fundamentals of Egress Peering Engineering](#) .

RELATED DOCUMENTATION

[Docker Overview](#)

[What is Docker?](#)

[What is a Container?](#)

[Get Started With Docker](#)

cRPD Resource Requirements

[Table 3 on page 21](#) lists the minimum resource requirements for cRPD.

Table 3: cRPD Minimum Resource Requirements

Description	Minimum Value
CPU	1 core
Memory	256 MB
Disk space	256 MB

cRPD Scaling

You can scale the performance and capacity of a cRPD by increasing the allocated amount of memory and the CPU available on the host hardware or VM resources.

[Table 4 on page 22](#) lists the cRPD scaling information,

Table 4: cRPD Scaling

Instance	RIB/FIB Route Scale	Minimum Memory
cRPD	32,000	256 MB
	64,000	512 MB
	1,28,000	1024 MB
	1,000,000	2048 MB

2

CHAPTER

Installing and Upgrading cRPD

Requirements for Deploying cRPD on a Linux Server | 25

Installing cRPD on Docker | 26

Upgrading cRPD | 33

Requirements for Deploying cRPD on a Linux Server

This section presents an overview of requirements for deploying a cRPD container on a Linux server:

Host Requirements

[Table 5 on page 25](#) lists the Linux host requirement specifications for deploying a cRPD container on a Linux server.

Table 5: Host Requirements

Component	Specification
Linux OS support	Ubuntu 18.04
Linux Kernel	4.15
Docker Engine	18.09.1
CPUs	1 CPU core
Memory	2 GB
Disk space	10 GB
Host processor type	x86_64 multicore CPU
Network Interface	Ethernet

Interface Naming and Mapping

[Table 6 on page 25](#) lists the supported interfaces on cRPD.

Table 6: Interface Naming and Mapping

Interface Number	cRPD Interfaces
eth0	eth0-mgmt-interface

Table 6: Interface Naming and Mapping (continued)

Interface Number	cRPD Interfaces
eth1	eth1-data-interface

Installing cRPD on Docker

IN THIS SECTION

- [Before You Install | 26](#)
- [Install and Verify Docker | 27](#)
- [Download the cRPD Software | 27](#)
- [Creating Data Volumes and Running cRPD using Docker | 28](#)
- [Configuring cRPD using the CLI | 29](#)
- [Configuring Memory | 33](#)

This section outlines the steps to install the cRPD container in a Linux server environment that is running Ubuntu or Red Hat Enterprise Linux (RHEL). The cRPD container is packaged in a Docker image and runs in the Docker Engine on the Linux host.

This section includes the following topics:

Before You Install

Before you install cRPD as routing service to achieve routing functionality in a Linux container environment, ensure to:

- Verify the system requirement specifications for the Linux server to deploy the cRPD, see [“Requirements for Deploying cRPD on a Linux Server” on page 25](#).

Install and Verify Docker

Install and configure Docker on Linux host platform to implement the Linux container environment, see [Install Docker](#) for installation instructions on the supported Linux host operating systems.

Verify the Docker installation. See [“Debugging cRPD Application” on page 97](#).

Download the cRPD Software

The cRPD software is available as a cRPD Docker file from the Juniper Internal Docker registry.

There are two ways to download the software:

- Juniper software download page
- Juniper Docker Registry

Before you import the cRPD software, ensure that Docker is installed on the Linux host and that the Docker Engine is running.

Once the Docker Engine has been installed on the host, perform the following to download and start using the cRPD image:

NOTE: You must include the **--privileged** flag in the **docker run** command to enable the cRPD container to run in privileged mode.

To download the cRPD software using the Juniper Docker Registry:

1. Log in to the Juniper Internal Docker registry using the login name and password that you received as part of the sales fulfillment process when ordering cRPD.

```
root@ubuntu-vm18:~# docker login hub.juniper.net -u <username> -p <password>
```

2. Pull the docker image from the download site using the following command:

```
root@ubuntu-vm18:~# docker pull hub.juniper.net/routing/crpd:19.2R1.8
```

3. Verify images in docker image repository.

```
root@ubuntu-vm18:~# docker images
```

REPOSITORY CREATED	TAG SIZE	IMAGE ID
crpd 6 days ago	19.2R1.8 278MB	4156a807054a

To download the cRPD software from the Juniper download URL:

1. Download the cRPD software image from the [Juniper Networks website](#).

```
root@ubuntu-vm18:~# docker load -i crpd-19.2R1.8.tgz
```

2. Verify the downloaded images in docker image repository

```
root@ubuntu-vm18:~# docker images
```

REPOSITORY CREATED	TAG SIZE	IMAGE ID
crpd 6 days ago	19.2R1.8 278MB	4156a807054a

Creating Data Volumes and Running cRPD using Docker

To create data volumes:

1. Create data volume for configuration and var logs.

```
root@ubuntu-vm18:~# docker volume create crpd01-config
```

```
crpd01-config
```

```
root@ubuntu-vm18:~# docker volume create crpd01-varlog
```

```
crpd01-varlog
```

Data volumes remain even after containers are destroyed and can be attached to newer containers. Data volumes are not shared between multiple containers at the same time unless they are ready-only volumes.

2. Attach the data volumes to create and launch the container to the cRPD instance.

```
root@ubuntu-vm18:~# docker run --rm --detach --name crpd01 -h crpd01 --net=bridge --privileged
-v crpd01-config:/config -v crpd01-varlog:/var/log -it crpd:19.2R1.8
```

Data volumes remain even after containers are destroyed and can be attached to newer containers. Data volumes are not shared between multiple containers at the same time unless they are ready-only volumes.

To launch cRPD in host networking mode:

1. Run the command to launch cRPD in host networking mode:

```
root@ubuntu-vm18:~# docker run --rm --detach --name crpd01 -h crpd01 --privileged --net=host -v
crpd01-config:/config -v crpd01-varlog:/var/log -it crpd:19.2R1.8
```

Configuring cRPD using the CLI

cRPD provides Junos command line configuration and operational commands for routing service. It provides subset of routing protocols configuration that enable node participates in topology and routing.

You can configure interfaces from Linux shell. Interface configuration is available only for the ISO addresses.

To configure the cRPD container using the CLI:

1. Log in to the cRPD container.

```
root@ubuntu-vm18:~/# docker exec -it crpd01 cli
```

2. Enter configuration mode.

```
root@crpd01> configure
```

```
Entering configuration mode
[edit]
```

3. Set the root authentication password by entering a cleartext password, an encrypted password, or an SSH public key string (DSA or RSA).

root@crpd01# set system root-authentication plain-text-password

```
New password: password
Retype new password: password
```

4. Commit the configuration to activate it on the cRPD instance.

root@crpd01# commit

```
commit complete
```

5. (Optional) Use the **show** command to display the configuration to verify that it is correct.

root@crpd01#show configuration

```
version 20190606.224121_builder.r1033375;
system {
    root-authentication {
        encrypted-password
"$6$JEC/p$QOUq12ew4tVJNKZYiCKT8CjnlP3SLu16BRIxvtz0CyBMc57WGu2oCyg/lTr0iR8oJMDumtEKi0HVo2NNFEJ.";
        ## SECRET-DATA
    }
}
```

root@crpd01# show

```
## Last changed: 2019-02-13 19:28:26 UTC
version "19.2I20190125_1733_rbu-builder [rbu-builder]";
routing-options {
    forwarding-table {
        export pplb;
    }
    router-id 90.90.90.20;
    autonomous-system 100;
}
protocols {
    bgp {
        group test {
            type internal;
            local-address 90.90.90.20;
            family inet {
                unicast;
```

```

    }
    neighbor 90.90.90.10 {
        bfd-liveness-detection {
            minimum-interval 100;
        }
    }
    neighbor 90.90.90.30 {
        bfd-liveness-detection {
            minimum-interval 100;
        }
    }
}
group test6 {
    type internal;
    local-address abcd::90:90:90:20;
    family inet6 {
        unicast;
    }
    neighbor abcd::90:90:90:10 {
        bfd-liveness-detection {
            minimum-interval 100;
        }
    }
    neighbor abcd::90:90:90:30 {
        bfd-liveness-detection {
            minimum-interval 100;
        }
    }
}
}
isis {
    level 1 disable;
    interface all {
        family inet {
            bfd-liveness-detection {
                minimum-interval 100;
            }
        }
        family inet6 {
            bfd-liveness-detection {
                minimum-interval 100;
            }
        }
    }
}

```

```

    }
    ospf {
        area 0.0.0.0 {
            interface all {
                bfd-liveness-detection {
                    minimum-interval 100;
                }
            }
        }
    }
    ospf3 {
        area 0.0.0.0 {
            interface all {
                bfd-liveness-detection {
                    minimum-interval 100;
                }
            }
        }
    }
}
policy-options {
    policy-statement pplb {
        then {
            load-balance per-packet;
        }
    }
}
interfaces {
    lo0 {
        unit 0 {
            family iso {
                address 49.0005.1111.2222.3333.00;
            }
        }
    }
}
}

```

Configuring Memory

To limit the amount of memory allocated to the cRPD:

1. You can specify the memory size using the following command:

```
root@ubuntu-vm18:~# docker run --rm --detach --name crpd01 -h crpd01 --privileged -v  
crpd01-config:/config -v crpd01-varlog:/var/log -m 2048MB --memory-swap=2048MB -it  
crpd:19.2R1.8
```

RELATED DOCUMENTATION

[Docker Engine User Guide](#)

[Docker Commands](#)

Upgrading cRPD

IN THIS SECTION

- [Upgrade Software](#) | 33

Upgrade Software

You can upgrade cRPD software by launching a new container using the newer image and attaching the persistent config volumes. Separate volumes are used to store config and logs. They are persistent even after the cRPD is stopped.

NOTE: cRPD does not support in place software upgrade.

To upgrade cRPD:

1. Ensure to import the latest cRPD image. See [“Installing cRPD on Docker” on page 26](#).

2. Stop the existing container.

```
docker stop crpd01
```

3. Run the container using latest version of cRPD.

```
docker run --rm --detach --name crpd01 -h crpd01 --privileged -v crpd01-config:/config -v  
crpd01-varlog:/var/log -m 2048MB --memory-swap=2048MB -it crpd:latest
```

RELATED DOCUMENTATION

| [Installing cRPD on Docker](#) | 26

3

CHAPTER

Managing cRPD

Managing cRPD | 37

Establishing an SSH Connection for a NETCONF Session and cRPD | 41

Managing cRPD

IN THIS SECTION

- [Viewing Container Processes in a Running cRPD | 38](#)
- [Accessing cRPD CLI and Bash Shell | 38](#)
- [Pausing and Resuming Processes within a cRPD Container | 39](#)
- [Removing a cRPD Instance | 39](#)
- [Viewing Docker Statistics and Logs | 40](#)
- [Viewing Active Containers | 41](#)
- [Stopping the Container | 41](#)

Viewing Container Processes in a Running cRPD

To view container processes:

1. Run the **docker exec** command to view the details about the processes (applications, services, and status) running on a container.

```
root@ubuntu-vm18:~# docker exec crpd01 ps aux
```

```
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  76996  8060 ?        Ss   Apr26    0:01 /sbin/init
root        19  0.0  0.2 160392 71520 ?        S<s  Apr26    0:38
/lib/systemd/systemd-journald
systemd+    30  0.0  0.0  70616  5236 ?        Ss   Apr26    0:00
/lib/systemd/systemd-resolved
root        32  0.0  0.0 167404 16324 ?        Ssl  Apr26    0:00 /usr/bin/python3
/usr/bin/networkd-dispatcher --run-startup-triggers
syslog      33  0.0  0.0  263036  4164 ?        Ssl  Apr26    0:05
/usr/sbin/rsyslogd -n
message+    38  0.0  0.0  49928  4072 ?        Ss   Apr26    0:00
/usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile
--systemd-activation --syslog-only
root        47  0.0  0.0  13020  1852 pts/0    Ss+  Apr26    0:00 /sbin/agetty
-o -p -- \u --noclear --keep-baud console 115200,38400,9600 xterm
root        52  0.0  0.0  72296  5536 ?        Ss   Apr26    0:00 /usr/sbin/sshd
-D
root        80  0.0  0.0 1453936 13584 ?        Ss   Apr26    0:01 /usr/sbin/mgd
-N
root        86  0.1  0.2 1053572 95040 ?        Ssl  Apr26    5:58 /usr/sbin/rpd
-N
root        87  0.0  0.0  837400  6356 ?        Ss   Apr26    0:01 /usr/sbin/ppmd
-N
root        88  0.0  0.0  842112  6460 ?        Ss   Apr26    0:01 /usr/sbin/bfdd
-N
root       102  0.0  0.0  13244  1832 tty1     Ss+  Apr26    0:00 /sbin/agetty
-o -p -- \u --noclear tty1 linux
root       108  0.0  0.0  18500  3340 pts/1    Ss   Apr26    0:00 /bin/bash
root       119  0.0  0.0  739724 11936 pts/1    S+   Apr26    0:02 cli
root       120  0.0  0.0 1454680 12636 ?        Ss   Apr26    0:00 /usr/sbin/mgd
-N
root      1502  0.0  0.0  34400  2704 ?        Rs   09:22    0:00 ps aux
```

Accessing cRPD CLI and Bash Shell

To access the cRPD using CLI and bash shell:

1. Run the **docker exec -it crpd1 cli** to launch the Junos CLI.

```
root@ubuntu-vm18:~# docker exec -it crpd01 cli
```

2. Run the **docker exec -it crpd1 bash** to launch the Junos CLI.

```
root@ubuntu-vm18:~# docker exec -it crpd01 bash
```

Pausing and Resuming Processes within a cRPD Container

You can pause or resume all processes within one or more containers.

To pause and restart a cRPD:

1. Run the **docker pause** command to suspend all the processes in a cRPD container.

```
root@ubuntu-vm18:~# docker pause crpd-container-name
```

2. Run the **docker unpause** command to resume all the processes in the cRPD container.

```
root@ubuntu-vm18:~# docker unpause crpd-container-name
```

Removing a cRPD Instance

To remove a cRPD instance or image:

NOTE: You must first stop and remove a cRPD instance before you remove a cRPD image.

1. Run the **docker stop** command to stop the cRPD.

```
root@ubuntu-vm18:~# docker stop crpd-container-name
```

```
crpd01
```

2. Run the **docker rm** command to remove the cRPD.

```
root@ubuntu-vm18:~# docker rm crpd-container-name
```

NOTE: Include **--force** to force the removal of the running cRPD.

3. Run the **docker rmi** command to remove one or more cRPD images from the Docker Engine.

NOTE: Include **--force** to force the removal a cRPD image.

```
root@ubuntu-vm18:~# docker rmi crd-Image-name
```

Viewing Docker Statistics and Logs

To view the statistics and logs:

1. Run the **docker logs** command for extracting the container logs.
2. Run the **docker stats** command to monitor the resource utilization.

Viewing Active Containers

To view the current active containers and their status:

1. Run the **docker ps** command to list the active containers.

```
root@ubuntu-vm18:~# docker container ls
```

```
root@ubuntu-vm18:~# docker ps
```

Stopping the Container

To stop the container:

1. You can stop the container using the following command:

```
root@ubuntu-vm18:~# docker stop crpd-container-name
```

RELATED DOCUMENTATION

| [Docker commands](#)

Establishing an SSH Connection for a NETCONF Session and cRPD

IN THIS SECTION

- [Establishing an SSH Connection | 42](#)
- [Enabling SSH | 42](#)
- [Port Forwarding Mechanism | 43](#)
- [Connecting to a NETCONF Server on Container | 43](#)

Establishing an SSH Connection

SSH can be used to establish connections between a *configuration management server* and a device running Linux OS with cRPD. A configuration management server, as the name implies, is used to configure the device running Linux OS remotely. With SSH, the configuration management server initiates an SSH session with the device running Linux OS.

Enabling SSH

To enable SSH on a cRPD:

1. Change the root password.

```
root@crpd01:/# sudo passwd root
```

```
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully
```

2. In the `/etc/ssh/sshd_config` file, specify the following setting:

```
root@crpd01:/# PermitRootLogin = yes
```

3. Restart the service.

```
root@crpd01:/#service ssh restart
```

```
* Starting OpenBSD Secure Shell server sshd  
  
[ OK ]
```


Port Forwarding Mechanism

To map a host port to a container port:

1. Run the following command to map a port on the host with a port on container.

```
root@crpd01:~# docker run -d --name crpd_instance -p host_port:container_port crpd:latest
```

```
root@crpd01:~# docker run -d --name crpd_instance -p 8034:22 crpd:latest
```

Connecting to a NETCONF Server on Container

1. Run the script to SSH to cRPD container using NETCONF:

```
root@crpd01:~# ssh root@<hostip> -p 8034 netconf
```

```
root@ubuntu-vm18's password:
```

```
Last login: Fri May  3 06:38:21 2019 from 10.223.4.154
```

```
===>
```

```
Containerized Routing Protocols Daemon (CRPD)
```

```
Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
```

```
<===
```

The 8034 port on host is mapped to 22 port on container and opens an interactive shell session.

4

CHAPTER

Programmable Routing

cRPD Application Development Using JET APIs | 47

Getting Started with JET | 47

cRPD Application Development Using JET APIs

JET is a framework that enables developers to create applications. cRPD supports JET which provides gRPC based routing APIs that are accessed locally from within the container or remotely over a TCP session. JET APIs support Programmable RPD.

JET APIs

cRPD supports the following JET Service APIs:

- **Routing:**
 - PRPD Common API
 - PRPD Service API
 - RIB Service API
 - MPLS Service API
 - Routing Interface Notification API
 - BGP Route Service API

For more information on the APIs, see [Juniper Extension Toolkit Developer Guide](#).

Getting Started with JET

IN THIS SECTION

- [Configure JET Interaction with Linux OS | 48](#)
- [Maximum Number of JET Connections | 48](#)
- [Compile IDL Files | 48](#)

Configure JET Interaction with Linux OS

In cRPD, JET service is enabled on TCP port 40051 and binds to loopback address 127.0.0.1 or ::1. To access JET service remotely, set up the SSH tunneling into the cRPD container using either username and password or SSH keys.

Remote access to JET service is secured using SSH. For more information on enabling SSH using port forwarding, see [“Establishing an SSH Connection for a NETCONF Session and cRPD” on page 41](#).

The maximum number of JET connections supported is 512.

Maximum Number of JET Connections

To setup maximum number of JET connections:

1. Access the cRPD Linux shell.
2. Use the command to add the connections:

```
root@crpd1:~# ulimit -n 519
```

3. Restart ssh on cRPD.

```
/etc/init.d/ssh restart
```

4. Re-establish ssh tunnel from Host Ubuntu to cRPD.
5. Connect up to 512 simultaneous JET Connections.

Compile IDL Files

To download and compile the IDL files:

1. Download the IDL file from the Juniper Networks website at www.juniper.net/support/downloads/.
2. Unpack the IDL file.

For example:

```
ubuntu-16:~ jet$ mkdir proto
ubuntu-16:~ jet$ tar -xzf jet-idl-18.4-20181107.0.tar.gz -C proto/
```

```
ubuntu-16:~ jet$ ls proto/
```

```
1 2 README
```

```
ubuntu-16:~ jet$ ls proto/2
```

```
jnx_authentication_service.proto
jnx_common_addr_types.proto
jnx_common_base_types.proto
jnx_management_service.proto
jnx_registration_service.proto
```

3. Compile python and gRPC modules for Authentication and Management Service proto files.

For example:

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_management_service.proto
```

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_authentication_service.proto
```

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_common_base_types.proto
```

```
ubuntu-16:~ jet$ ls -lrt
```

```
total 112
-rw-r--r-- 1 vagrant vagrant 52683 Nov  8 16:47 jet-idl-18.4-20181107.0.tar.gz
drwxr-xr-x 1 vagrant vagrant  170 Nov  8 16:49 proto
-rw-r--r-- 1 vagrant vagrant 40924 Nov  8 16:56 jnx_management_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  4719 Nov  8 16:56
jnx_management_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  5365 Nov  8 2018 jnx_authentication_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  1898 Nov  8 2018
jnx_authentication_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  6391 Nov  8 2018 jnx_common_base_types_pb2.py
-rw-r--r-- 1 vagrant vagrant    83 Nov  8 2018
jnx_common_base_types_pb2_grpc.py
```

For details on how to generate code from an IDL file in the language of your choice, see <https://www.grpc.io/docs>.

5

CHAPTER

Configuring cRPD Features

Configuring Settings on Host OS | **53**

Multitopology Routing in cRPD | **59**

Layer 3 Overlay Support in cRPD | **66**

MPLS Support in cRPD | **80**

Configuring Settings on Host OS

IN THIS SECTION

- [Configuring ARP Scaling | 53](#)
- [Configuring OSPFv2/v3 | 54](#)
- [Configuring MPLS | 54](#)
- [Adding MPLS Routes | 55](#)
- [Adding Routes with MPLS label | 55](#)
- [Creating a VRF device | 56](#)
- [Assigning a Network Interface to a VRF | 56](#)
- [Viewing the Devices assigned to VRF | 57](#)
- [Viewing Neighbor Entries to VRF | 57](#)
- [Viewing Addresses for a VRF | 57](#)
- [Viewing Routes for a VRF | 58](#)
- [Removing Network Interface from a VRF | 59](#)

This chapter provides information on tuning of settings on host OS to enable advanced features or to increase the scale of cRPD functionality.

Configuring ARP Scaling

cRPD uses a static route to forward traffic for routes destined to interfaces. You need to create a static route and specify the next-hop address.

To configure the cRPD container in static routing mode:

1. Launch the cRPD container:

```
root@crpd-ubuntu3:~/crpd# docker exec -it crpd01 -- /bin/bash
```

2. Log in and configure static routes.

```
root@crpd# cli
```

```
root@crpd> configure
```

3. View the forwarding table to verify the static routes.

```
root@crpd> show route forwarding-table
```

4. The maximum ARP entry number is controlled by the Linux host kernel. If there are a large number of neighbors, you might need to adjust the ARP entry limitations on the Linux host. There are options in the **sysctl** command on the Linux host to adjust the ARP or NDP entry limits.

For example, to adjust the maximum ARP entries using IPv4:

```
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh1=4096
```

```
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh2=8192
```

```
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh3=8192
```

For example, to adjust the maximum ND entries using IPv6:

```
root@host:~# sysctl -w net.ipv6.neigh.default.gc_thresh1=4096
```

```
root@host:~# sysctl -w net.ipv6.neigh.default.gc_thresh2=8192
```

```
root@host:~#sysctl -w net.ipv6.neigh.default.gc_thresh3=8192
```

Configuring OSPFv2/v3

To allow more number of OSPFv2/v3 adjacencies on cRPD:

1. Increase the IGMP membership limit.

```
root@host:~# sysctl -w net.ipv4.igmp_max_memberships=1000
```

Configuring MPLS

To configure MPLS:

1. Load the MPLS modules in the container using **modprobe** or **insmod** :

```
root@crpd-ubuntu3:~# modprobe mpls_ip tunnel
```

```
root@crpd-ubuntu3:~# modprobe mpls_router
```

```
root@crpd-ubuntu3:~# modprobe iptunnel
```

2. Verify the MPLS modules loaded in host OS.

```
root@host:~# lsmod | grep mpls
```

```
mpls_iptunnel      16384  0
mpls_router        28672  1 mpls_iptunnel
ip_tunnel          24576  1 mpls_router
```

3. After loading the **mpls_router** on the host, configure the following commands to activate MPLS on the interface.

```
root@host:~#sudo sysctl -w net.mpls.platform_labels=1048575
```

Adding MPLS Routes

To add MPLS routes to host using the **iproute2** utility:

1. Run the following command to add the mpls routes to the host OS.

```
root@host:~#ip -f mpls route add 100 as 200/300 via inet 192.0.2.2 dev swp1
```

2. Run the following command to view the mpls routes.

```
root@host:~#ip -f mpls route show
```

```
100 as to 200/300 via inet 192.0.2.2 dev swp1
```

Adding Routes with MPLS label

To add routes to host by encapsulating the packets with MPLS label using the **iproute2** utility:

1. Run the following command to encapsulate the packets to host OS.

```
root@host:~#ip route add 192.0.2.1/30 encap mpls 200 via inet 192.1.1.1 dev swp1
```

2. Run the following command to view the mpls routes.

```
root@host:~#ip route show
```

```
192.0.2.1/30 encap mpls 200 via 192.1.1.1 dev swp1
```

Creating a VRF device

To instantiate a VRF device and associate it with a table:

1. Run the following command to create a VRF device.

```
root@host:~#ip link add dev NAME type vrf table ID
```

2. Run the following command to view the created VRFs.

```
root@host:~#ip [-d] link show type vrf
```

3. Run the following command to view the list of VRFs in the host OS.

```
root@host:~#ip vrf show
```

Assigning a Network Interface to a VRF

Network interfaces are assigned to a VRF by enslaving the netdevice to a VRF device. On enslavement, connected and local routes are automatically moved to the table associated with the VRF device.

To assign a network interface to a VRF:

1. Run the following command to assign a interface.

```
root@host:~#ip link set dev <name> master <name>
```

```
root@host:~#ip link set dev eth0 master mgmt
```

Viewing the Devices assigned to VRF

To view the devices:

1. Run the following command to view the devices assigned to a VRF.

```
root@host:~#ip link show vrf <name>
```

```
root@host:~#ip link show vrf red
```

```
root@host:~#ip link show master <NAME>
```

Viewing Neighbor Entries to VRF

To list the neighbor entries associated with devices enslaved to a VRF device:

1. Run the following command to add the master option to the ip command:

```
root@host:~#ip [-6] neigh show vrf <NAME>
```

```
root@host:~#ip neigh show vrf red
```

```
root@host:~#ip -6 neigh show vrf red
```

```
root@host:~#ip [-6] neigh show master <NAME>
```

Viewing Addresses for a VRF

To show addresses for interfaces associated with a VRF:

1. Run the following command to add the master option to the ip command:

```
root@host:~#ip addr show vrf <NAME>
```

```
root@host:~#ip addr show vrf red
```

```
root@host:~#ip addr show master <NAME>
```

Viewing Routes for a VRF

To view routes for a VRF:

1. Run the following command to view the IPv6 routes table associated with the VRF device:

```
root@host:~# ip [-6] route show vrf NAME
```

```
root@host:~# ip [-6] route show table ID
```

2. Run the following command to do a route lookup for a VRF device:

```
root@host:~# ip [-6] route get vrf <NAME> <ADDRESS>
```

```
root@host:~# ip route get 192.0.2.1 vrf red
```

```
root@host:~# ip [-6] route get oif <NAME> <ADDRESS>
```

```
root@host:~# ip -6 route get 2001:db8::32 vrf red
```

```
2001:db8::32 from :: dev eth1 table red proto kernel src 2001:1::2 metric
256 pref medium
```

3. Run the following command to view the IPv4 routes in a VRF device:

```
root@host:~# Ip route list table <table-id>
```

```
root@host:~# Ip route list table <table-id>
```

Removing Network Interface from a VRF

Network interfaces are removed from a VRF by breaking the enslavement to the VRF device

1. Run the following command to remove the network interface:

```
root@host:~# ip link set dev NAME nomaster
```

After removing the network interface, connected routes are moved to the default table and local entries are moved to the local table.

RELATED DOCUMENTATION

[Example: Configuring Static Label Switched Paths for MPLS in cRPD | 81](#)

[Example: Configuring Layer 3 VPN \(VRF\) on cRPD Instance | 68](#)

Multitopology Routing in cRPD

IN THIS SECTION

- [Understanding Multitopology in cRPD | 59](#)
- [Example: Configuring Multitopology Routing with BGP in cRPD | 60](#)

Understanding Multitopology in cRPD

cRPD enables BGP multiple RIBs functionality to support Multitopology routing (MTR) based on the routing policy with Linux FIBs (routes in forwarding plane). The applications can select required routing table based on the routing policy from the Linux FIB in cRPD for different types of traffic. Each type of traffic is defined by a topology that is used to create a new routing table for that topology. Each topology uses the unified control plane to make routing decisions for traffic associated with that topology. In addition, each topology has a separate forwarding table and, in effect, a dedicated forwarding plane for each topology.

Service providers and enterprises can use multitopology routing (MTR) to engineer traffic flow across a network. MTR can be used with direct and static routes, IS-IS, OSPF, and BGP. In a network carrying

multiple traffic types, you often need to direct different types of application traffic over multiple links depending on their link characteristics. Communities are used for BGP when exporting routes to multitopology. OSPFv3 does not support MTR. MTR discovers IGP routes and able to resolve BGP routes against the custom topologies with static and OSPF. .

You can configure separate topologies to share the same network links as needed. MTR uses a combination of control plane (routing) and forwarding plane filters.

MTR provides the ability to generate forwarding tables based on the resolved entries in the routing tables for the topologies you create. MTR and forwarding is available only on master routing instance. A dedicated RIB is created for storing the Multitopology routes. BGP multipath is not enabled on topologies.

When routing topologies are configured under **routing-options**, a new routing table for each topology is created. Each routing protocol creates a routing table based on the topology name, the instance name, and the purpose of the table.

Example: Configuring Multitopology Routing with BGP in cRPD

IN THIS SECTION

- [Requirements | 60](#)
- [Overview | 60](#)
- [Configuration | 61](#)
- [Verification | 64](#)

This example shows how to configure community based multiple topologies with BGP in cRPD and unicast the traffic using MTR over network paths.

Requirements

This example requires following software release:

- cRPD 19.4R1 or later.

Overview

Multi-topology support for BGP is based on the community value in a BGP route. This configuration determines the association between a topology and one or more community values and populates the

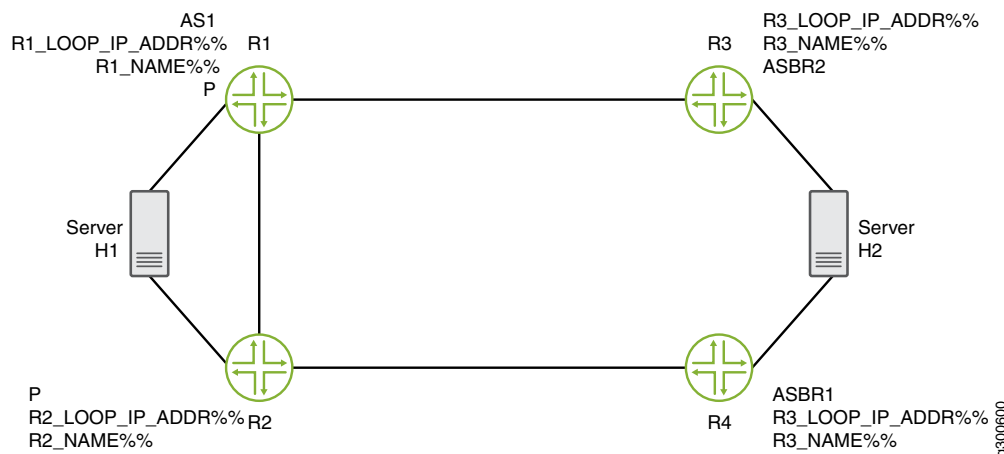
topology routing tables. Arriving BGP updates that have a matching community value are 1647 replicated in the associated topology routing table.

Configure topologies with BGP **inet** family and verify BGP import matching route into topology rib. For each topology a list of community objects must be provided such that the routing software can setup an internal **ribgroup** and the corresponding secondary table import policy.

Topology

Figure 5 on page 61 shows topology for configuring multi-topology with BGP.

Figure 5: Multi-topology Routing



Configuration

IN THIS SECTION

- [Configuring BGP through Multitopology Routing | 62](#)
- [Results | 63](#)

To configure multitopology with BGP:

CLI Quick Configuration

```

set routing-options topologies family inet topology red table-id 40
set routing-options topologies family inet topology blue table-id 41
set routing-options topologies family inet topology green table-id 42

```

```

set routing-options router-id 2.2.2.2
set routing-options autonomous-system 500
set routing-options rib :red.inet.0 static route 1.1.1.1/32 next-hop 1.15.0.2
set routing-options rib :green.inet.0 static route 1.1.1.1/32 next-hop 1.13.0.2
set routing-options rib :blue.inet.0 static route 1.1.1.1/32 next-hop 1.17.0.2
set protocols bgp group ibgp-app-rr-ser type internal
set protocols bgp group ibgp-app-rr-ser traceoptions file bgp size 100m
set protocols bgp group ibgp-app-rr-ser traceoptions flag update
set protocols bgp group ibgp-app-rr-ser traceoptions flag state
set protocols bgp group ibgp-app-rr-ser local-address 77.1.1.1
set protocols bgp group ibgp-app-rr-ser family inet unicast add-path send path-count 6
set protocols bgp family inet unicast topology red community 1:1
set protocols bgp family inet unicast topology green community 1:2
set protocols bgp family inet unicast topology blue community 1:3

```

Configuring BGP through Multitopology Routing

Step-by-Step Procedure

1. Configure multiple topologies.

```

[edit routing-options topologies]
set family inet topology red table-id 40
set family inet topology blue table-id 41
set family inet topology green table-id 42

```

2. Configure static routes.

```

[edit routing-options]
user@crpd# set router-id 2.2.2.2
user@crpd# set autonomous-system 500
user@crpd# set rib :red.inet.0 static route 1.1.1.1/32 next-hop 1.15.0.2
user@crpd# set rib :green.inet.0 static route 1.1.1.1/32 next-hop 1.13.0.2
user@crpd# set rib :blue.inet.0 static route 1.1.1.1/32 next-hop 1.17.0.2

```

3. Configure BGP group parameters to import matching route into topology rib. BGP uses the target community identifier to install the routes it learns in the appropriate multi-topology routing tables.

```

[edit protocols bgp]
set group ibgp-app-rr-ser type internal
set group ibgp-app-rr-ser traceoptions file bgp size 100m
set group ibgp-app-rr-ser traceoptions flag update
set group ibgp-app-rr-ser traceoptions flag state

```

```

set group ibgp-app-rr-ser local-address 77.1.1.1
set group ibgp-app-rr-ser family inet unicast add-path send path-count 6
set family inet unicast topology red community 1:1
set family inet unicast topology green community 1:2
set family inet unicast topology blue community 1:3

```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp** and **show routing-options** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```

show routing-options
topologies {
  family inet {
    topology red {
      table-id 40;
    }
    topology blue {
      table-id 41;
    }
    topology green {
      table-id 42;
    }
  }
}
rib :red.inet.0 {
  static {
    route 1.1.1.1/32 next-hop 1.15.0.2;
  }
}
rib :green.inet.0 {
  static {
    route 1.1.1.1/32 next-hop 1.13.0.2;
  }
}
rib :blue.inet.0 {
  static {
    route 1.1.1.1/32 next-hop 1.17.0.2;
  }
}

```

```

user@crpd#show protocols bgp
family inet {
  unicast {
    topology red {
      community 1:1;
    }
    topology green {
      community 1:2;
    }
    topology blue {
      community 1:3;
    }
  }
  group ibgp-app-rr-ser {
    type internal;
    traceoptions {
      file bgp size 100m;
      flag update;
    }
    local-address 77.1.1.1;
    family inet {
      unicast {
        add-path {
          send {
            path-count 6;
          }
        }
      }
    }
  }
}

```

If you are done configuring the device, enter commit from configuration mode.

Verification

Verifying BGP routes

Purpose

To verify BGP matched routes:

Action

From operational mode, enter the **show route protocol bgp all table** command:

user@crpd> **show route protocol bgp all table**

```
:red.inet.0: 11 destinations, 11 routes (8 active, 0 holddown, 3 hidden)
+ = Active Route, - = Last Active, * = Both

99.9.9.1/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

99.9.9.2/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

99.9.9.5/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

:green.inet.0: 10 destinations, 10 routes (8 active, 0 holddown, 2 hidden)
+ = Active Route, - = Last Active, * = Both

99.9.9.1/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

99.9.9.4/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

:blue.inet.0: 11 destinations, 11 routes (8 active, 0 holddown, 3 hidden)
+ = Active Route, - = Last Active, * = Both

99.9.9.3/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

99.9.9.4/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1

99.9.9.5/32      [BGP/170] 00:05:07, localpref 100, from 128.49.114.118
                AS path: I, validation-state: unverified
                > to 1.15.0.2 via ens4f1
```

From operational mode, enter the **show route protocol bgp all table inet.0** command:

user@crpd> **show route protocol bgp all table inet.0**

```
inet.0: 20 destinations, 20 routes (20 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

99.9.9.1/32      *[BGP/170] 00:00:14, localpref 100, from 128.49.114.118
                  AS path: I, validation-state: unverified
                  > to 1.15.0.2 via ens4f1
99.9.9.2/32      *[BGP/170] 00:00:14, localpref 100, from 128.49.114.118
                  AS path: I, validation-state: unverified
                  > to 1.15.0.2 via ens4f1
99.9.9.3/32      *[BGP/170] 00:00:14, localpref 100, from 128.49.114.118
                  AS path: I, validation-state: unverified
                  > to 1.15.0.2 via ens4f1
99.9.9.4/32      *[BGP/170] 00:00:14, localpref 100, from 128.49.114.118
                  AS path: I, validation-state: unverified
                  > to 1.15.0.2 via ens4f1
99.9.9.5/32      *[BGP/170] 00:00:14, localpref 100, from 128.49.114.118
                  AS path: I, validation-state: unverified
                  > to 1.15.0.2 via ens4f1
```

Meaning

You can view BGP matching routes installed to RIB tables and when the routes without community target are available only in inet.0 table.

SEE ALSO

Understanding Multitopology Routing

Understanding Multitopology Routing for Class-Based Forwarding of Voice, Video, and Data Traffic

Layer 3 Overlay Support in cRPD

IN THIS SECTION

- [Understanding Layer 3 Overlay VRF support in cRPD | 67](#)
- [Example: Configuring Layer 3 VPN \(VRF\) on cRPD Instance | 68](#)

Understanding Layer 3 Overlay VRF support in cRPD

Starting in Junos OS Release 19.4R1, virtual routing and forwarding (VRF) instances are supported in cRPD along with the support of MPLS and Multiprotocol BGP to provide overlay functionality.

A routing instance is a collection of routing tables, interfaces, and routing protocol parameters. To implement Layer 3 VPNs, you configure one routing instance for each VPN. A VRF is a network device in the Linux kernel and the device is associated with **table-id**. You configure the routing instances on PE routers only. You can create VRFs in the Linux network. VRF device implementation impacts only Layer 3 and above. Each VPN routing instance consists of the following components:

- VRF table—On each PE router, you configure one VRF table for each VPN.
- Policy rules—These control the import of routes into and the export of routes from the VRF table.
- One or more routing protocols that install routes from CE routers into the VRF table—You can use the BGP, OSPF, and RIP routing protocols, and you can use static routes.

When a VRF device is created, it is associated with a routing table. Packets that come in through enslaved devices to the VRF are looked up in the routing table associated with the VRF device. Similarly egress routing rules are used to send packets to the VRF driver before sending it out on the actual interface.

VRF is used to manage routes and to forward traffic based on independent forwarding tables in VRF. RPD creates multiple routing tables for every routing instance of type **vrf**. The tables are one for each address family. You need to configure a routing instance for each VPN on each of the PE routers participating in the VPN. You can configure routing instances using the **[edit routing-instances]** hierarchy. The routing instance of type **vrf** is only supported on cRPD.

You can create multiple instances of BFD, BGP, IS-IS, OSPF version 2 (referred as OSPF), OSPF version 3 (OSPFv3), and ICMP router discovery under a VRF using the **[edit routing-instances routing-instance-name protocols]** hierarchy. You can configure protocol independent routing using the **edit routing-instances instance-name routing-options** hierarchy.

Layer-3 Overlay supports the following tunneling protocols in cRPD:

- Static routes in inet.3
- BGP labeled unicast
- GRE tunneling
- MPLS static LSPs
- Routes programmed using programmable-rpd APIs
- direct-ebgp-peering on MPLS enabled interface

Moving the Interfaces under a VRF

The enslavement of devices is done by RPD that is interfaces configured under the routing instance are migrated (enslaved) to the vrf-device by RPD using a netlink message sent to the kernel.

When an interface is configured under the routing instance of type **vrf**, if such a link has been learnt from the kernel and the link is not associated to the correct table, RPD sends a netlink notification to enslave the link. If the link does not exist or RPD has not learnt about the link, whenever the link is created or RPD learns about it then the link will be enslaved correctly based on the configuration.

Example: Configuring Layer 3 VPN (VRF) on cRPD Instance

IN THIS SECTION

- Requirements | 68
- Overview | 69
- Configuration | 69
- Verification | 76

This example shows the VPNv4 route resolution on PE routers and route reflectors by configuring the PE routers with specific policies to control the import of routes into and the export of routes from the VRF table and with next hops learnt using BGP labeled unicast. In this example, the traffic flows from CE1 to CE2.

Requirements

This example uses the following hardware and software components:

- Ubuntu software version 18.04
- Linux kernel version 4.5 or later
- cRPD software Release version 19.4R1 or later

Before you configure a Layer 3 VPN (VRF), you must install the basic components:

- MPLS modules on the host OS on which the cRPD instance is created. For details, see [“Configuring Settings on Host OS” on page 53](#).

- Provider edge router (PE1), a provider router (P), and provider edge router (PE2). For installing, see [“Installing cRPD on Docker” on page 26](#).

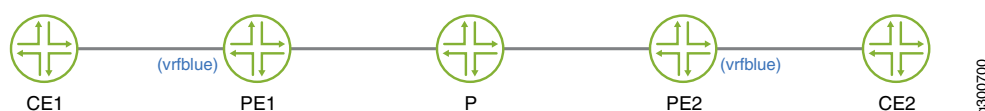
Overview

To configure the VPNv4 route resolution, you need to configure a routing instance of type VRF for each VPN on each of the PE routers participating in the VPN and add static routes to it. The **static** statement configures the static routes that are installed in the **vrfblue.inet.0** routing table. There is no loopback interface or device for every VRF device created in the Linux kernel. But the loopback host addresses are directly added to the VRF device which can be learnt by RPD.

Topology

Figure 6 on page 69 shows the Layer 3 VPN (VRF) Topology

Figure 6: Layer 3 VPN (VRF) Topology



Configuration

Configuring PE1 router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```
user@crpd1# set routing-options rib mpls.0
```

2. Configure policy that accepts routes.

```
[edit policy-options policy-statement]
user@crpd1# set EXPORT_LO term 10 from route-filter 2.2.2.2/32 exact
user@crpd1# set EXPORT_LO term 10 then accept
user@crpd1# set NH_SELF term 10 then next-hop self
```

3. Configure a VRF routing instance on PE1 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd1# set routing-options static route 1.1.1.1/32 next-hop 10.10.10.1
user@crpd1# set instance-type vrf
user@crpd1# set route-distinguisher 100:100
user@crpd1# set vrf-target target:100:100
```

4. Configure the router ID.

```
user@crpd1# set routing-options router-id 2.2.2.2
```

5. Configure BGP session.

```
[edit protocols bgp group]
user@crpd1# set underlay type external family inet unicast
user@crpd1# set underlay type external export EXPORT_LO neighbor 20.20.20.3 family inet labeled-unicast
resolve-vpn
user@crpd1# set underlay type external export EXPORT_LO neighbor 20.20.20.3 peer-as 2 local-as 1
user@crpd1# set VPN type internal local-address 2.2.2.2 family inet-vpn unicast
user@crpd1# set VPN local-as 5
user@crpd1# set VPN neighbor 4.4.4.4 family inet-vpn unicast
```

6. Configure the interface on MPLS.

```
user@crpd1# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp** and **show routing-instances** commands. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd1# show routing-instances
vrfblue {
  routing-options {
    static {
      route 1.1.1.1/32 next-hop 10.10.10.1;
    }
  }
  instance-type vrf;
  route-distinguisher 100:100;
```

```

    vrf-target target:100:100;
}
user@crpd1# show protocols bgp
group underlay {
    type external;
    family inet {
        unicast;
    }
    export EXPORT_LO;
    neighbor 20.20.20.3 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 2;
        local-as 1;
    }
    neighbor 20.20.20.2 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 1;
        local-as 2;
    }
    neighbor 30.30.30.4 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 3;
        local-as 4;
    }
}
group VPN {
    type internal;
    local-address 2.2.2.2;
    family inet-vpn {
        unicast;
    }
    local-as 5;
}

```

```
neighbor 4.4.4.4 {
  family inet-vpn {
    unicast;
  }
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring P router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```
user@crpd2# set routing-options rib mpls.0
```

2. Configure policy that accepts routes.

```
[edit policy-options policy-statement]
user@crpd2# set EXPORT_LO term 10 from route-filter 3.3.3.3/32 exact
user@crpd2# set EXPORT_LO term 10 then accept
user@crpd2# set NH_SELF term 10 then next-hop self
```

3. Configure BGP session.

```
[edit protocols bgp group]
user@crpd2# set underlay type external export EXPORT_LO neighbor 20.20.20.2 family inet labeled-unicast
resolve-vpn
user@crpd2# set underlay type external export EXPORT_LO neighbor 20.20.20.2 peer-as 1
user@crpd2# set underlay type external export EXPORT_LO neighbor 20.20.20.2 local-as 2
user@crpd2# set underlay type external export EXPORT_LO neighbor 30.30.30.4 family inet labeled-unicast
resolve-vpn
user@crpd2# set underlay type external export EXPORT_LO neighbor 30.30.30.4 peer-as 3
user@crpd2# set underlay type external export EXPORT_LO neighbor 30.30.30.4 local-as 4
```

4. Configure the router ID.

```
user@crpd2# set routing-options router-id 3.3.3.3
```

5. Configure the interface on MPLS.

```
user@crpd2# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp** and **show policy-options** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@crpd2# show protocols bgp
group underlay {
  type external;
  export EXPORT_LO;
  neighbor 20.20.20.2 {
    family inet {
      labeled-unicast {
        resolve-vpn;
      }
    }
  }
  peer-as 1;
  local-as 2;
}
neighbor 30.30.30.4 {
  family inet {
    labeled-unicast {
      resolve-vpn;
    }
  }
  peer-as 3;
  local-as 4;
}
}
```

```
user@crpd2# show policy-options
policy-statement EXPORT_LO {
  term 10 {
    from {
      route-filter 3.3.3.3/32 exact;
    }
    then accept;
  }
}
```

```

policy-statement NH_SELF {
  term 10 {
    then {
      next-hop self;
    }
  }
}

```

Configuring PE2 router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```

user@crpd3# set routing-options rib mpls.0

```

2. Configure policy that accepts routes.

```

[edit policy-options policy-statement]
user@crpd3# set EXPORT_LO term 10 from route-filter 4.4.4.4/32 exact
user@crpd3# set EXPORT_LO term 10 then accept
user@crpd3# set NH_SELF term 10 then next-hop self

```

3. Configure a VRF routing instance on PE2 and other routing instance parameters.

```

[edit routing-instances vrfblue]
user@crpd3# set routing-options static route 5.5.5.5/32 next-hop 40.40.40.5
user@crpd3# set instance-type vrf
user@crpd3# set route-distinguisher 100:100
user@crpd3# set vrf-target target:100:100
user@crpd3# set interface all

```

4. Configure BGP session.

```

[edit protocols bgp group]
user@crpd3# set underlay type external export EXPORT_LO neighbor 30.30.30.3 family inet labeled-unicast
resolve-vpn
user@crpd3# set underlay type external export EXPORT_LO neighbor 30.30.30.3 peer-as 4
user@crpd3# set underlay type external export EXPORT_LO neighbor 30.30.30.3 local-as 3
user@crpd3# set VPN type internal local-address 4.4.4.4 family inet-vpn unicast

```

```
user@crpd3# set VPN local-as 5
user@crpd3# set VPN neighbor 2.2.2.2 family inet-vpn unicast
```

5. Configure the router ID.

```
user@crpd3# set routing-options router-id 4.4.4.4
```

6. Configure the interface on MPLS.

```
user@crpd3# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp** and **show routing-instances** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@crpd3# show protocols bgp
group underlay {
  export EXPORT_LO;
  neighbor 30.30.30.3 {
    family inet {
      labeled-unicast {
        resolve-vpn;
      }
    }
    peer-as 4;
    local-as 3;
  }
}
group VPN {
  type internal;
  local-address 4.4.4.4;
  family inet-vpn {
    unicast;
  }
  local-as 5;
  neighbor 2.2.2.2 {
    family inet-vpn {
      unicast;
    }
  }
}
```

```
}
}
```

```
user@crpd3# show routing-instances
vrflblue {
  routing-options {
    static {
      route 5.5.5.5/32 next-hop 40.40.40.5;
    }
  }
  interface all;
  instance-type vrf;
  route-distinguisher 100:100;
  vrf-target target:100:100;
}
```

Verification

IN THIS SECTION

- [Verifying VPNv4 Resolution on PE1 | 76](#)
- [Verifying BGP LU on P | 77](#)
- [Verifying VPNv4 Resolution on PE2 | 78](#)

Verifying VPNv4 Resolution on PE1

Purpose

To verify VPNv4 routes on PE1:

Action

From operational mode, enter the **show route table vrflblue.inet.0 5.5.5.5** command:

```
user@crpd1>show route table vrflblue.inet.0 5.5.5.5
```

```
vrflblue.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```



```

5.5.5.5/32          *[BGP/170] 00:00:14, localpref 100, from 4.4.4.4
                    AS path: I, validation-state: unverified
                    > to 20.20.20.3 via pe1-p, Push 299808, Push 299792(top)

```

From operational mode, enter the **show route table mpls.0** command:

```
user@crpd1>show route table mpls.0
```

```

mpls.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

299808              *[VPN/170] 00:01:45
                    > to 10.10.10.1 via pe1-ce1, Pop
299808(S=0)         *[VPN/170] 00:01:45
                    > to 10.10.10.1 via pe1-ce1, Pop
299824              *[VPN/170] 00:01:45
                    receive table vrfblue.inet.0, Pop

```

From bash mode, enter the **ip route list table 5 5.5.5.5** command:

```
user@crpd1>ip route list table 5 5.5.5.5
```

```
5.5.5.5 encap mpls 299792/299808 via 20.20.20.3 dev pe1-p proto 22
```

From bash mode, enter the **ip -f mpls route** command:

```
user@crpd1>ip -f mpls route
```

```
299808 via inet 10.10.10.1 dev pe1-ce1 proto 22
```

Meaning

You can view PE1 has a route under **vrfblue.inet.0** to CE2 which is learnt from PE2 with nexthop 4.4.4.4, which is resolved using BGP LU from P router.

Verifying BGP LU on P

Purpose

To verify VPNv4 routes on P:

Action

From bash mode, enter the **ip -f mpls route show** command:

```
user@crpd2>ip -f mpls route show
```

```
299776 via inet 20.20.20.2 dev p-pe1 proto 22
299792 via inet 30.30.30.4 dev p-pe2 proto 22
```

From operational mode, enter the **show route table mpls.0** command:

```
user@crpd2>show route table mpls.0
```

```
mpls.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0          *[MPLS/0] 01:40:42, metric 1
           Receive
1          *[MPLS/0] 01:40:42, metric 1
           Receive
2          *[MPLS/0] 01:40:42, metric 1
           Receive
13         *[MPLS/0] 01:40:42, metric 1
           Receive
299776     *[VPN/170] 01:19:24
           > to 20.20.20.2 via p-pe1, Pop
299776(S=0) *[VPN/170] 01:19:24
           > to 20.20.20.2 via p-pe1, Pop
299792     *[VPN/170] 01:19:20
           > to 30.30.30.4 via p-pe2, Pop
299792(S=0) *[VPN/170] 01:19:20
           > to 30.30.30.4 via p-pe2, Pop
```

Meaning

You can view the MPLS and VPN routes from P to PE1 and P to PE2.

Verifying VPNv4 Resolution on PE2

Purpose

To verify VPNv4 routes on PE2:

Action

From operational mode, enter the **show route table vrfblue.inet.0 1.1.1.1** command:

user@crpd3>**show route table vrfblue.inet.0 1.1.1.1**

```
vrfblue.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.1.1/32          *[BGP/170] 00:00:26, localpref 100, from 2.2.2.2
                    AS path: I, validation-state: unverified
                    > to 30.30.30.3 via pe2-p, Push 299808, Push 299776(top)
```

From operational mode, enter the **show route table mpls.0** command:

user@crpd3>**show route table mpls.0**

```
mpls.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 01:34:39, metric 1
                   Receive
1                  *[MPLS/0] 01:34:39, metric 1
                   Receive
2                  *[MPLS/0] 01:34:39, metric 1
                   Receive
13                 *[MPLS/0] 01:34:39, metric 1
                   Receive
299808             *[VPN/170] 00:00:43
                   > to 40.40.40.5 via pe2-ce2, Pop
299808(S=0)        *[VPN/170] 00:00:43
                   > to 40.40.40.5 via pe2-ce2, Pop
299824             *[VPN/170] 00:00:43
                   receive table vrfblue.inet.0, Pop
```

From bash mode, enter the **ip route list table 5 1.1.1.1** command:

user@crpd3>**ip route list table 5 1.1.1.1**

```
1.1.1.1 encap mpls 299776/299808 via 30.30.30.3 dev pe2-p proto 22
```

From bash mode, enter the **ip -f mpls route** command:

user@crpd3>**ip -f mpls route**

```
299808 via inet 40.40.40.5 dev pe2-ce2 proto 22
```

Meaning

On PE2 router, PE1 displays the routes for the VRF table **vrfblue.inet.0** using BGP LU about 1.1.1.1 as a VPNv4 prefix with nexthop as 2.2.2.2..

SEE ALSO

vrf-target

vrf-import

route-distinguisher

vrf-export

vrf-table-label

no-vrf-advertise

Routing Instances in Layer 3 VPNs

MPLS Support in cRPD

IN THIS SECTION

- [Understanding MPLS support in cRPD | 80](#)
- [Example: Configuring Static Label Switched Paths for MPLS in cRPD | 81](#)

Understanding MPLS support in cRPD

Multiprotocol Label Switching (MPLS) configuration is supported in cRPD for forwarding packets to the destination in MPLS network.

With MPLS, only the first device does a routing lookup. Instead of finding the next hop, the device finds the ultimate destination along with a path to that destination. The path of an MPLS packet is called a label-switched path (LSP). LSPs are unidirectional routes through a network or an autonomous system

(AS). MPLS routers within an AS determine paths through a network through the exchange of MPLS traffic engineering information. Using these paths, the routers direct traffic through the network along an established route. Rather than selecting the next hop along the path as in IP routing, each router is responsible for forwarding the packet to a predetermined next hop address.

Routers that are part of the LSP are label-switching routers (LSRs). An MPLS LSP is established using static LSPs. A static LSP requires each router along the path to be configured explicitly. You must manually configure the path and its associated label values.

cRPD supports only a limited number of Junos OS MPLS features. You can configure MPLS **interface**, **ipv6-tunneling**, **label-history**, **label-range**, and **static-label-switched-path** in cRPD CLI under the **edit protocols mpls** hierarchy.

Supported Features

- BGP configuration
- MPLS using PRPD API
- BGP labeled unicast configuration

SEE ALSO

<i>mpls</i>
<i>static-label-switched-path</i>
Example: Configuring Static Label Switched Paths for MPLS in cRPD 81
Configuring Settings on Host OS 53

Example: Configuring Static Label Switched Paths for MPLS in cRPD

IN THIS SECTION

- [Requirements | 82](#)
- [Overview | 82](#)
- [Configuration | 83](#)
- [Verification | 87](#)

This example shows how the VPN traffic flows through a v4 MPLS tunnel among PEs by configuring BGP and MPLS static label switched paths.

Requirements

This example uses the following hardware and software components:

- Ubuntu software version 18.04
- Linux kernel version 4.5 or later
- cRPD software Release version 19.4R1 or later

Before you configure a static LSP for MPLS forwarding, you must install the basic components:

- MPLS modules on host OS on which cRPD instance is created. For details, see [“Configuring Settings on Host OS” on page 53](#).
- Provider edge router (PE1), a provider router (P), and provider edge router (PE2). For installing, see [“Installing cRPD on Docker” on page 26](#).

Overview

In this example, PE1 acts as a Label Edge Router or ingress node to the MPLS network, which encapsulates the packets by attaching labels. P acts as Label Switching Router that transfers MPLS packets using labels in the MPLS network.

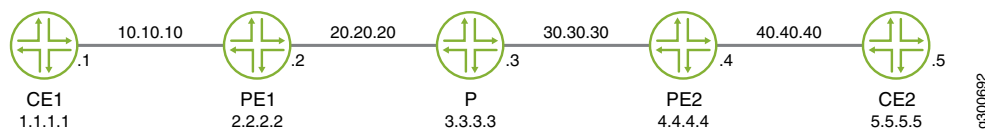
To configure MPLS, you must first create one or more named paths on the ingress and transit routers. For each path, you can specify some or all transit routers in the path.

Configuring static label-switched paths (LSPs) for MPLS is similar to configuring static routes on individual routers.

Topology

[Figure 7 on page 82](#) shows the topology used in this example.

Figure 7: MPLS Forwarding in cRPD



Configuration

IN THIS SECTION

- [Configuring PE1 Router | 83](#)
- [Configuring Provider P Router. | 85](#)
- [Configuring PE2 Router | 86](#)
- [Results | 87](#)

To configure static LSP for MPLS on cRPD:

Configuring PE1 Router

Step-by-Step Procedure

To configure the static LSP:

1. Create the tables inet.0 and mpls.0.

```
[edit routing-options]
user@crpd1# set rib inet.0
user@crpd1# set rib mpls.0
user@crpd1# set router-id 2.2.2.2
```

2. Configure BGP session.

```
[edit protocols bgp group VPN]
user@crpd1# set type internal local-address 2.2.2.2 family inet-vpn unicast
user@crpd1# set local-as 5
user@crpd1# set neighbor 4.4.4.4 family inet-vpn unicast
```

3. Configure the static label range and ingress static LSP parameters.

```
[edit protocols mpls]
user@crpd1# set interface all
user@crpd1# set label-range static-label-range 1000000 1048575
user@crpd1# set static-label-switched-path pe2 ingress install 4.4.4.4/32 active
user@crpd1# set static-label-switched-path pe2 ingress to 4.4.4.4 next-hop 20.20.20.2 push 1000001
```

4. Configure a static route from the ingress PE2.

```
[edit routing-options static]
user@crpd1# set route 2.2.2.2/32 next-hop 20.20.20.2
user@crpd1# set route 4.4.4.4/32 static-lsp-next-hop pe2
```

5. Configure a VRF routing instance on PE1 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd1# set routing-options static route 1.1.1.1/32 next-hop 10.10.10.1
user@crpd1# set route-distinguisher 100:100
user@crpd1# set vrf-target target:100:100
user@crpd1# set interface all
```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp** and **run show configuration protocols mpls** commands on PE1. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd1# show protocols bgp
group VPN {
  type internal;
  local-address 2.2.2.2;
  family inet-vpn {
    unicast;
  }
  local-as 5;
  neighbor 4.4.4.4 {
    family inet-vpn {
      unicast;
    }
  }
}
```

```
user@crpd1# run show configuration protocols mpls
interface all;
static-label-switched-path pe2 {
  ingress {
    next-hop 20.20.20.3;
    to 4.4.4.4;
    push 1000001;
```



```
}
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring Provider P Router.

Step-by-Step Procedure

To configure the static LSP:

1. Configure router ID for router P.

```
[edit routing-options]
user@crpd2# set rib mpls.0
user@crpd2# set router-id 3.3.3.3
```

2. Configure a transit static LSP for swap and pop labels.

```
[edit protocols mpls]
user@crpd2# set label-range static-label-range 1000000 1048575
user@crpd2# set static-label-switched-path pe2 transit 1000001 next-hop 30.30.30.4 swap 1000002
user@crpd2# set static-label-switched-path pe1 transit 1000003 next-hop 20.20.20.2 swap 1000004
user@crpd2# set static-label-switched-path pe2 transit 1000001 pop next-hop 30.30.30.4
user@crpd2# set static-label-switched-path pe1 transit 1000003 pop next-hop 20.20.20.2
```

Results

From configuration mode, confirm your configuration by entering the **show protocols bgp**, **run show configuration protocols mpls**, and **run show mpls interface** commands on P. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd2# run show configuration protocols mpls
interface all;
static-label-switched-path pe1 {
  transit 1000003 {
    next-hop 20.20.20.2;
    swap 1000004;
  }
}
static-label-switched-path pe2 {
  transit 1000001 {
    next-hop 30.30.30.4;
    swap 1000002;
```

```
}
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring PE2 Router

Step-by-Step Procedure

To configure the static LSP for MPLS on PE2:

1. Configure BGP session.

```
[edit protocols bgp group VPN ]
user@crpd3# set type internal local-address 4.4.4.4 family inet-vpn unicast
user@crpd3# set local-as 5
user@crpd3# set neighbor 2.2.2.2 family inet-vpn unicast
```

2. Configure the ingress static LSP parameters.

```
[edit protocols mpls ]
user@crpd3# set interface all
user@crpd3# set label-range static-label-range 1000000 1048575
user@crpd3# set static-label-switched-path pe1 ingress install 2.2.2.2/32 active
user@crpd3# set static-label-switched-path pe1 ingress to 2.2.2.2 next-hop 30.30.30.4 push 1000003
```

3. Configure router ID and a static route from the ingress PE1.

```
[edit routing-options]
user@crpd3# set rib inet.0
user@crpd3# set router-id 4.4.4.4
user@crpd3# set static route 4.4.4.4/32 next-hop 30.30.30.4
user@crpd3# set static route 2.2.2.2/32 static-lsp-next-hop pe1
```

4. Configure a VRF routing instance on PE2 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd3# set routing-options static route 5.5.5.5/32 next-hop 40.40.40.5
user@crpd3# set route-distinguisher 100:100
user@crpd3# set vrf-target target:100:100
user@crpd3# set interface all
```

Results

From configuration mode, confirm your configuration by entering the **run show configuration protocols mpls** and **run show mpls interface** commands on PE2. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd3# show protocols bgp
group VPN {
  type internal;
  local-address 4.4.4.4;
  family inet-vpn {
    unicast;
  }
  local-as 5;
  neighbor 2.2.2.2 {
    family inet-vpn {
      unicast;
    }
  }
}
```

```
user@crpd3# run show configuration protocols mpls
interface all;
static-label-switched-path pe2 {
  ingress {
    next-hop 20.20.20.3;
    to 4.4.4.4;
    push 1000001;
  }
}
```

If you are done configuring the device, enter commit from configuration mode.

Verification

Verify MPLS forwarding on PE1

Purpose

To verify the configuration for MPLS on PE1.

Action

From operational mode, enter the **show route table vrfblue.inet.0 5.5.5.5** command:

```
user@crpd1> show route table vrfblue.inet.0 5.5.5.5
```

```

vrfblue.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

5.5.5.5/32          *[BGP/170] 00:01:03, localpref 100, from 4.4.4.4
                    AS path: I, validation-state: unverified
                    > to 20.20.20.3 via pe1-p, Push 299776, Push 1000001(top)

```

From operational mode, enter the **show mpls label usage** command:

```
user@crpd1> show mpls label usage
```

```

Label space Total    Available      Applications
LSI           999984  999983 (100.00%) BGP/LDP VPLS with no-tunnel-services, BGP
L3VPN with vrf-table-label
Block         999984  999983 (100.00%) BGP/LDP VPLS with tunnel-services, BGP L2VPN
Dynamic       999984  999983 (100.00%) RSVP, LDP, PW, L3VPN, RSVP-P2MP, LDP-P2MP,
MVPN, EVPN, BGP
Static        48576   48576  (100.00%) Static LSP, Static PW
Effective Ranges
Range name    Shared with Start    End
Dynamic       16      999999
Static        1000000 1048575
Configured Ranges
Range name    Shared with Start    End
Dynamic       16      999999
Static        1000000 1048575

```

From operational mode, enter the **show mpls static-lsp** command:

```
user@crpd1>show mpls static-lsp
```

```

Ingress LSPs:
LSPname              To              State
pe2                  4.4.4.4         Up
Total 1, displayed 1, Up 1, Down 0

Transit LSPs:
Total 0, displayed 0, Up 0, Down 0

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

```

```
Segment LSPs:
Total 0, displayed 0, Up 0, Down 0
```

From operational mode, enter the **show route table inet.3** command:

```
user@crpd1> show route table inet.3
```

```
inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

4.4.4.4/32          *[MPLS/6/1] 00:04:44, metric 0
                    > to 20.20.20.3 via pe1-p, Push 1000001
```

From operational mode, enter the **show route table mpls.0** command:

```
user@crpd1> show route table mpls.0
```

```
mpls.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 00:15:45, metric 1
                   Receive
1                  *[MPLS/0] 00:15:45, metric 1
                   Receive
2                  *[MPLS/0] 00:15:45, metric 1
                   Receive
13                 *[MPLS/0] 00:15:45, metric 1
                   Receive
299776             *[VPN/170] 00:06:32
                   > to 10.10.10.1 via pe1-ce1, Pop
299776(S=0)        *[VPN/170] 00:06:32
                   > to 10.10.10.1 via pe1-ce1, Pop
```

From operational mode, enter the **ip route list table 5 5.5.5.5** command:

```
user@crpd1> ip route list table 5 5.5.5.5
```

```
5.5.5.5  encap mpls  1000001/299776 via 20.20.20.3 dev pe1-p proto 22
```

From operational mode, enter the **ip -f mpls route** command:

```
user@crpd1> ip -f mpls route
```

```
299776 via inet 10.10.10.1 dev pe1-ce1 proto 22
```

Verify MPLS forwarding on P

Purpose

To verify the configuration for MPLS on P.

Action

From shell mode, enter the **show route table mpls.0** command:

```
user@crpd2>show route table mpls.0
```

```
mpls.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0          *[MPLS/0] 00:00:11, metric 1
            Receive
1          *[MPLS/0] 00:00:11, metric 1
            Receive
2          *[MPLS/0] 00:00:11, metric 1
            Receive
13         *[MPLS/0] 00:00:11, metric 1
            Receive
299776     *[VPN/170] 00:00:05
            > to 20.20.20.2 via p-pe1, Pop
299776(S=0) *[VPN/170] 00:00:05
            > to 20.20.20.2 via p-pe1, Pop
299792     *[VPN/170] 00:00:05
            > to 30.30.30.4 via p-pe2, Pop
299792(S=0) *[VPN/170] 00:00:05
            > to 30.30.30.4 via p-pe2, Pop
1000001    *[MPLS/6] 00:00:11, metric 1
            > to 30.30.30.4 via p-pe2, Swap 1000002
1000003    *[MPLS/6] 00:00:11, metric 1
            > to 20.20.20.2 via p-pe1, Swap 1000004
```

```
user@crpd2>show mpls static-lsp
```

```

Ingress LSPs:
Total 0, displayed 0, Up 0, Down 0

Transit LSPs:
LSPname                Incoming-label  State
pe1                     1000003        Up
pe2                     1000001        Up
Total 2, displayed 2, Up 2, Down 0

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

Segment LSPs:
Total 0, displayed 0, Up 0, Down 0

```

From bash shell mode, enter the **ip -f mpls route** command:

```
user@crpd2:/# ip -f mpls route
```

```

299776 via inet 20.20.20.2 dev p-pe1 proto 22
299792 via inet 30.30.30.4 dev p-pe2 proto 22
1000001 as to 1000002 via inet 30.30.30.4 dev p-pe2 proto 22
1000003 as to 1000004 via inet 20.20.20.2 dev p-pe1 proto 22

```

Verify MPLS forwarding on PE2

Purpose

To verify the configuration for MPLS on P.

Action

From shell mode, enter the **show route table vrflblue.inet.0 1.1.1.1** command:

```
user@crpd3>show route table vrflblue.inet.0 1.1.1.1
```

```

vrflblue.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.1.1/32          *[BGP/170] 00:03:00, localpref 100, from 2.2.2.2

```

```
AS path: I, validation-state: unverified
> to 30.30.30.3 via pe2-p, Push 299776, Push 1000003(top)
```

user@crpd3>show mpls static-lsp

```
Ingress LSPs:
LSPname                To                State
pe1                    2.2.2.2          Up
Total 1, displayed 1, Up 1, Down 0

Transit LSPs:
LSPname                Incoming-label    State
pe2                    1000002          Dn
Total 1, displayed 1, Up 0, Down 1

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

Segment LSPs:
Total 0, displayed 0, Up 0, Down 0
```

user@crpd3>show route table mpls.0

```
mpls.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                *[MPLS/0] 00:17:31, metric 1
                  Receive
1                *[MPLS/0] 00:17:31, metric 1
                  Receive
2                *[MPLS/0] 00:17:31, metric 1
                  Receive
13               *[MPLS/0] 00:17:31, metric 1
                  Receive
299776           *[VPN/170] 00:03:07
                  > to 40.40.40.5 via pe2-ce2, Pop
299776(S=0)      *[VPN/170] 00:03:07
                  > to 40.40.40.5 via pe2-ce2, Pop
```

From bash shell mode, enter the **ip -f mpls route** command:


```
user@crpd3:/# ip -f mpls route
```

```
299776 via inet 40.40.40.5 dev pe2-ce2 proto 22
```

From bash shell mode, enter the **ip route list table 5 1.1.1.1** command:

```
user@crpd3:/# ip route list table 5 1.1.1.1
```

```
1.1.1.1 encap mpls 1000003/299776 via 30.30.30.3 dev pe2-p proto 22
```

Meaning

You can verify the static LSP between PEs are up on all the devices and the routes are populated in the corresponding route tables **inet.o** and **inet.3** and in the Linux FIB.

SEE ALSO

mpls

static-label-switched-path

6

CHAPTER

Troubleshooting

Debugging cRPD Application | 97

Debugging cRPD Application

IN THIS SECTION

- [Command-Line Interface | 97](#)
- [Fault Handling | 98](#)
- [Troubleshooting Container | 98](#)
- [Verify Docker | 99](#)
- [Viewing Core Files | 100](#)
- [Configuring Syslog | 100](#)

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

Command-Line Interface

The Junos OS command-line interface (CLI) is the primary tool for controlling and troubleshooting router hardware, the Junos OS, routing protocols, and network connectivity. CLI commands display information from routing tables, information specific to routing protocols, and information about network connectivity derived from the **traceroute** utilities. RPD tracelog facilities are supported and enabled through the CLI. Trace log files are stored **/var/log** path.

You can use the following Junos CLI commands to troubleshoot cRPD:

- **show task**: Display the routing protocol tasks on the Routing Engine.
- **show task memory detail**: Display the memory utilization for routing protocol tasks on the Routing Engine.
- **show route**: Display the active entries in the routing tables.
- **show bfd**: Display information about active Bidirectional Forwarding Detection (BFD) sessions.
- **show bgp**: Display information about BGP summary information for all routing instances.
- **show (ospf | ospf3)**: Display standard information about all OSPF neighbors for all routing instances.
- **show interfaces routing**: Perform router diagnostics.

- **show log:** View system activity logs and allows you to monitor and view information for performance monitoring, troubleshooting, and debugging purposes.
- **show krt:** Monitor KRT queues and their states.
- **show programmable-rpd:** List clients connected to the programmable routing protocol process (prpd) server. The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create, and modify the behavior of their network.
- **ip monitor:** Monitor the installation of routes to Linux FIB and interface events and netlink messages.
- **tcpdump:** Capture network traffic to/from control plane.
- **netstat:** Monitor the sockets.
- **request support information:** Display the support information which is used for troubleshooting.

Fault Handling

When the rpd crashes due to some issue, the rpd process is restarted automatically. To recover manually from a fault, you can implement the following CLI command hierarchies to handle the faults:

- **restart routing:** Restart the rpd.
- **clear bgp:** Clear BGP sessions.
- **deactivate:** Deactivate CLI configuration.
- **activate:** Activate the CLI configuration.

Troubleshooting Container

You can implement various docker commands to monitor and troubleshoot issues at container level when cRPD is deployed as a docker container.

- **docker ps:** List out active containers and their state.
- **docker stats:** Continuous monitor the resource utilization.
- **docker logs:** Extract container logs in case the container terminates unexpectedly.
- **docker stop:** Stop the Docker from the current state.
- **docker start:** Restart the Docker container.

Verify Docker

1. Verify the installed Docker Engine version by using the **docker version** command.

```
root@ubuntu-vm18:~# docker version
```

```
Client:
  Version:           18.09.1
  API version:       1.39
  Go version:        go1.10.6
  Git commit:        4c52b90
  Built:             Wed Jan  9 19:35:31 2019
  OS/Arch:           linux/amd64
  Experimental:      false

Server: Docker Engine - Community
Engine:
  Version:           18.09.1
  API version:       1.39 (minimum version 1.12)
  Go version:        go1.10.6
  Git commit:        4c52b90
  Built:             Wed Jan  9 19:02:44 2019
  OS/Arch:           linux/amd64
  Experimental:      false
```

2. View the software and hardware information in the system.

```
root@ubuntu-vm18:~# uname -a
```

```
Linux ubuntu-vm18 4.15.0-43-generic #46-Ubuntu SMP Thu Dec 6 14:45:28 UTC 2018
x86_64 x86_64 x86_64 GNU/Linux
```

3. View the version of ubuntu.

```
root@ubuntu-vm18:~# lsb_release -a
```

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
```

Viewing Core Files

Purpose

When a core file is generated, you can find the output at **/var/crash**. The core files generated are stored on the system that is hosting the Docker containers.

You can also use `ping` and `ping6` to check the reachability at the shell mode.

Action

To list the core files:

1. Exit from the CLI environment to return to the host unix shell.

```
user@host> start shell
```

2. Change the directory to **/var/crash**:

```
root@ubuntu-vm18$ cd /var/crash
```

```
root@ubuntu-vm18$ ls -l
```

3. Run the command to identify the location of the core files:

```
root@ubuntu-vm18$ sysctl kernel.core_pattern
```

4. Verify for any core files created around the time of the crash.

Configuring Syslog

Syslog is enabled by default and the messages are stored at **/var/log/messages** file stored on the local Routing Engine.

To configure remote syslog:

1. Access the cRPD Linux shell.
2. Open the **/etc/rsyslog.conf** file.
3. Add the following facility information:

```
*.* @<IP address>:<port>
```

Where: **<IP address>** is the IP address of the remote syslog server.

4. Save the file.

5. Restart syslog by using the following command:

```
root@crpd1# service rsyslog restart
```

To view the log messages:

1. You can view the log messages using the following command:

```
root@crpd1> show log messages
```

SEE ALSO

| *Log File Sample Content*