Juniper Cloud Native Router 25.2 User Guide

Published
2025-07-02

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

Juniper Cloud Native Router 25.2 User Guide

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

Table of Contents

1

CHAPTER

Introduction

IN THIS CHAPTER

Juniper Cloud-Native Router Overview

SUMMARY

This topic provides an overview of the Juniper Cloud-Native Router (JCNR) overview, use cases, and features.

IN THIS SECTION

Overview

While 5G unleashes higher bandwidth, lower latency and higher capacity, it also brings in new infrastructure challenges such as increased number of base stations or cell sites, more backhaul links with larger capacity and more cell site routers and aggregation routers. Service providers are integrating cloud-native infrastructure in distributed RAN (D-RAN) topologies, which are usually small, leased spaces, with limited power, space and cooling. The disaggregation of radio access network (RAN) and the expansion of 5G data centers into cloud hyperscalers has added newer requirements for cloud-native routing.

The Juniper Cloud-Native Router provides the service providers the flexibility to roll out the expansion requirements for 5G rollouts, reducing both the CapEx and OpEx.

Juniper Cloud-Native Router (JCNR) is a containerized router that combines Juniper's proven routing technology with the Junos containerized routing protocol daemon (cRPD) as the controller and a high-performance Data Plane Development Kit (DPDK) or extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based vRouter forwarding plane. It is implemented in Kubernetes and interacts seemlessly with a Kubernetes container network interface (CNI) framework.

Use Cases

The Cloud-Native Router has the following use cases:

- **Radio Access Network (RAN)**

  The new 5G-only sites are a mix of centralized RAN (C-RAN) and distributed RAN (D-RAN). The C-RAN sites are typically large sites owned by the carrier and continue to deploy physical routers. The D-RAN sites, on the other hand, are tens of thousands of smaller sites, closer to the users.

Optimization of CapEx and OpEx is a huge factor for the large number of D-RAN sites. These sites are also typically leased, with limited space, power and cooling capacities. There is limited connectivity over leased lines for transit back to the mobile core. Juniper Cloud-Native Router is designed to work in the constraints of a D-RAN. It is integrated with the distributed unit (DU) and installable on an existing 1 U server.

- **Telco virtual private cloud (VPC)**

  The 5G data centers are expanding into cloud hyperscalers to support more radio sites. The cloud-native routing available in public cloud environments do not support the routing demands of telco VPCs, such as MPLS, quality of service (QoS), L3 VPN, and more. The Juniper Cloud-Native Router integrates directly into the cloud as a containerized network function (CNF), managed as a cloud-native Kubernetes component, while providing advanced routing capabilities.

Architecture and Key Components

The Juniper Cloud-Native Router consists of the Junos containerized routing protocol Daemon (cRPD) as the control plane (Cloud-Native Router Controller), providing topology discovery, route advertisement and forwarding information base (FIB) programming, as well as dynamic underlays and overlays. It uses the Data Plane Development Kit (DPDK) or eBPF XDP datapath enabled vRouter as a forwarding plane, providing packet forwarding for applications in a pod and host path I/O for protocol sessions. The third component is the Cloud-Native Router container network interface (CNI) that interacts with Kubernetes as a secondary CNI to create pod interfaces, assign addresses and generate the router configuration.

The Data Plane Development Kit (DPDK) is an open source set of libraries and drivers. DPDK enables fast packet processing by allowing network interface cards (NICs) to send direct memory access (DMA) packets directly into an application's address space. The applications poll for packets, to avoid the overhead of interrupts from the NIC. Integrating with DPDK allows a vRouter to process more packets per second than is possible when the vRouter runs as a kernel module.

The extended Berkley Packet Filter (eBPF) is a Linux kernel technology that executes user-defined programs inside a sandbox virtual machine. It enables low-level networking programs to execute with optimal performance. The eXpress Data Path (XDP) frameworks enables high-speed packet processing for the eBPF programs. Cloud-Native Router supports eBPF XDP datapath based vRouter.

In this integrated solution, the Cloud-Native Router Controller uses gRPC, a high performance Remote Procedure Call, based services to exchange messages and to communicate with the vRouter, thus creating the fully functional Cloud-Native Router. This close communication allows you to:

- Learn about fabric and workload interfaces.
- Provision DPDK or kernel-based interfaces for Kubernetes pods as needed.
- Configure IPv4 and IPv6 address allocation for pods.

- Run routing protocols such as ISIS, BGP, and OSPF and much more.

Features

- Easy deployment, removal, and upgrade on general purpose compute devices using Helm.
- Higher packet forwarding performance with DPDK-based JCNR-vRouter.
- Full routing, switching, and forwarding stacks in software.
- Out-of-the-box software-based open radio access network (O-RAN) support.
- Quick spin up with containerized deployment.
- Highly scalable solution.
- L3 features such as transit gateway, support for routing protocols, BFD, VRRP, VRF-Lite, EVPN Type-5, ECMP and BGP Unnumbered, access control lists, SRv6.
- L2 functionality, such as MAC learning, MAC aging, MAC limiting, native VLAN, L2 statistics, and access control lists (ACLs).
- L2 reachability to Radio Units (RU) for management traffic.
- L2 or L3 reachability to physical distributed units (DU) such as 5G millimeter wave DUs or 4G DUs.
- VLAN tagging and bridge domains.
- Trunk and access ports.
- Support for multiple virtual functions (VF) on Ethernet NICs.
- Support for bonded VF interfaces.
- Rate limiting of egress broadcast, unknown unicast, and multicast traffic on fabric interfaces.
- IPv4 and IPv6 routing.

Juniper Cloud-Native Router Components

SUMMARY

The Juniper Cloud-Native Router solution consists of several components including the Cloud-Native Router controller, the Data Plane Development Kit (DPDK) or extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based Cloud-Native Router vRouter and the JCNR-CNI. This topic provides a brief overview of the components of the Juniper Cloud-Native Router.

IN THIS SECTION

Cloud-Native Router Components

The Juniper Cloud-Native Router has primarily three components—the Cloud-Native Router Controller control plane, the Cloud-Native Router vRouter forwarding plane, and the JCNR-CNI for Kubernetes integration. All Cloud-Native Router components are deployed as containers.

Figure 1 on page 6 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster when implemented with DPDK based vRouter.

Figure 1: Components of Juniper Cloud-Native Router (DPDK Datapath)

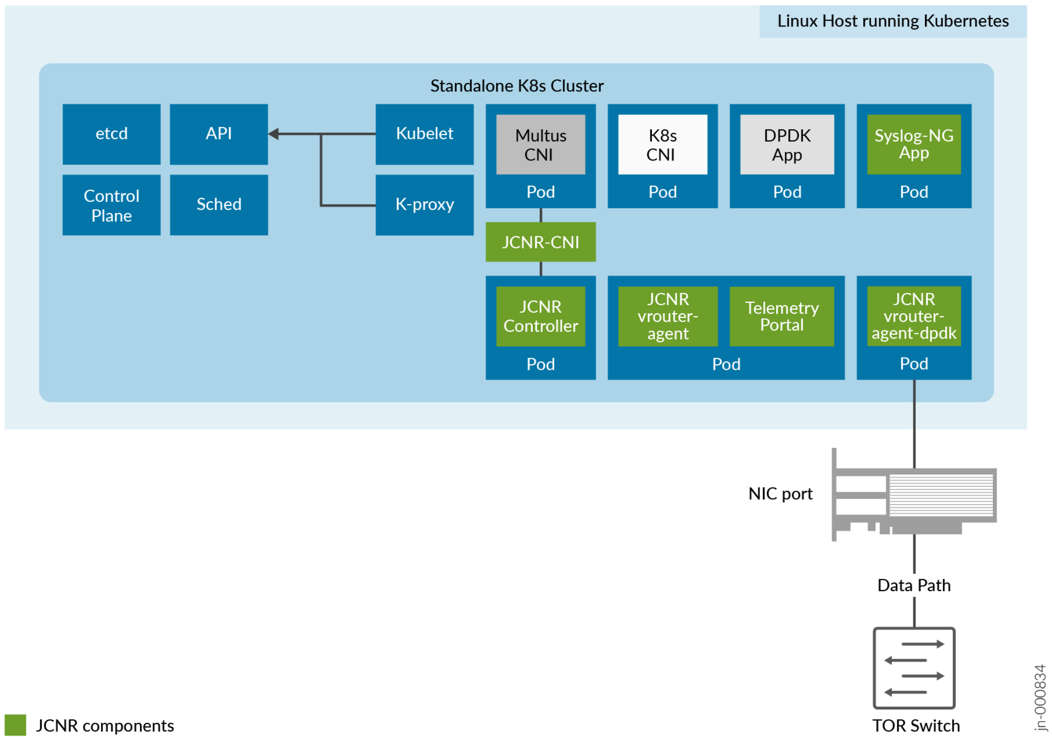
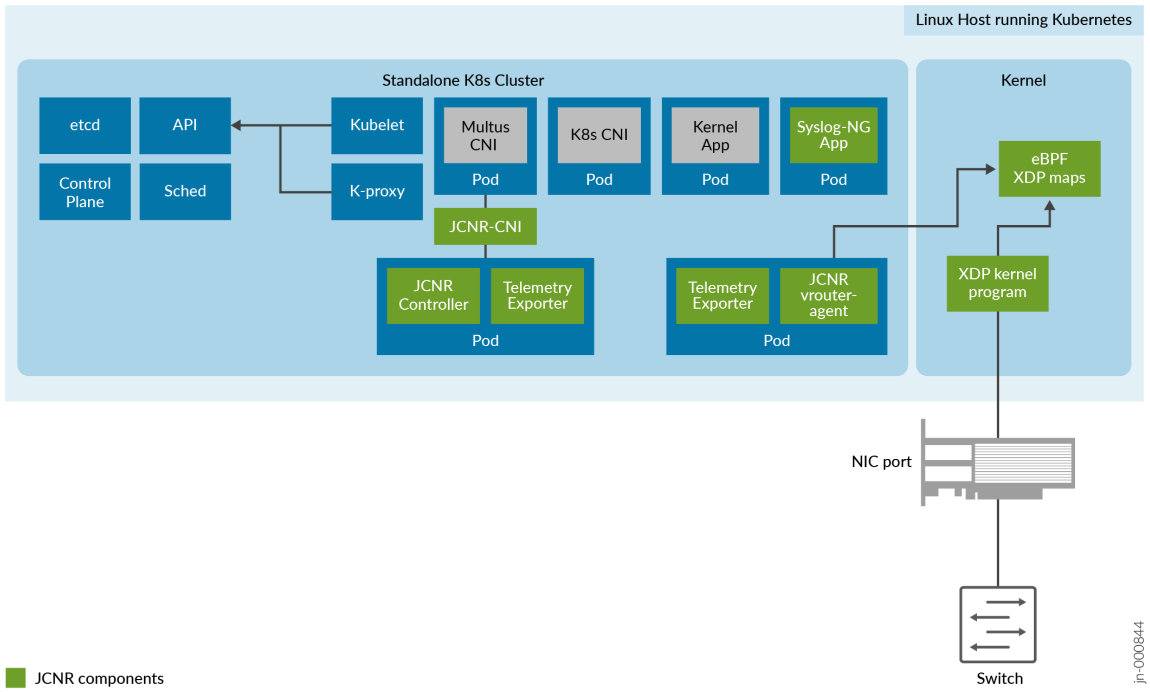


Figure 2 on page 7 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster when implemented with eBPF XDP based vRouter.

Figure 2: Components of Juniper Cloud-Native Router (eBPF XDP Datapath)



Cloud-Native Router Controller

The Cloud-Native Router Controller is the control-plane of the cloud-native router solution that runs the Junos containerized routing protocol Daemon (cRPD). It is implemented as a statefulset. The controller communicates with the other elements of the cloud-native router. Configuration, policies, and rules that you set on the controller at deployment time are communicated to the Cloud-Native Router vRouter and other components for implementation.

For example, firewall filters (ACLs) configured on the controller are sent to the Cloud-Native Router vRouter (through the vRouter agent).

Juniper Cloud-Native Router Controller Functionality:

- Exposes Junos OS compatible CLI configuration and operation commands that are accessible to external automation and orchestration systems using the NETCONF protocol.

- Supports vRouter as the high-speed forwarding plane. This enables applications that are built using the DPDK framework to send and receive packets directly to the application and the vRouter without passing through the kernel.

- Supports configuration of VLAN-tagged sub-interfaces on physical function (PF), virtual function (VF), virtio, access, and trunk interfaces managed by the DPDK-enabled vRouter.
- Supports configuration of bridge domains, VLANs, and virtual-switches.
- Advertises DPDK application reachability to core network using routing protocols primarily with BGP, IS-IS and OSPF.
- Distributes L3 network reachability information of the pods inside and outside a cluster.
- Maintains configuration for L2 firewall.
- Passes configuration information to the vRouter through the vRouter-agent.
- Stores license key information.
- Works as a BGP Speaker, establishing peer relationships with other BGP speakers to exchange routing information.
- Exports control plane telemetry data to Prometheus and gNMI.

Configuration Options

Use the *configlet resource* to configure the cRPD pods.

Cloud-Native Router vRouter

The Cloud-Native Router vRouter is a high-performance datapath component. It is an alternative to the Linux bridge or the Open vSwitch (OVS) module in the Linux kernel. It runs as a user-space process. The vRouter functionality is implemented in two pods, one for the vrouter-agent and the vrouter-telemetry-exporter, and the other for the vrouter-agent-dpdk. This split gives you the flexibility to tailor CPU resources to the different vRouter components as needed.

The vRouter supports both Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath.

> **NOTE**: Cloud-Native Router eBPF XDP Datapath is a "Juniper Technology Preview (Tech Preview)" on page 333 feature. Limited features are supported. See *Juniper Cloud-Native Router vRouter Datapath* for more details.

Cloud-Native Router vRouter Functionality:

- Performs routing with Layer 3 virtual private networks.

- Performs L2 forwarding.
- Supports high-performance DPDK-based forwarding.
- Supports high performance eBPF XDP datapath based forwarding.
- Exports data plane telemetry data to Prometheus and gNMI.

Benefits of vRouter:

- High-performance packet processing.
- Forwarding plane provides faster forwarding capabilities than kernel-based forwarding.
- Forwarding plane is more scalable than kernel-based forwarding.
- Support for the following NICs:
  - Intel E810 (Columbiaville) family
  - Intel XL710 (Fortville) family
  - NVIDIA Mellanox ConnectX-6 and ConnectX-7

JCNR-CNI

JCNR-CNI is a new container network interface (CNI) developed by Juniper. JCNR-CNI is a Kubernetes CNI plugin installed on each node to provision network interfaces for application pods. During pod creation, Kubernetes delegates pod interface creation and configuration to JCNR-CNI. JCNR-CNI interacts with Cloud-Native Router controller and the vRouter to setup DPDK interfaces. When a pod is removed, JCNR-CNI is invoked to de-provision the pod interface, configuration, and associated state in Kubernetes and cloud-native router components. JCNR-CNI works as a secondary CNI, along with the Multus CNI to add and configure pod interfaces.

JCNR-CNI Functionality:

- Manages the networking tasks in Kubernetes pods such as:
  - assigning IP addresses.
  - allocating MAC addresses.
  - setting up untagged, access, and other interfaces between the pod and vRouter in a Kubernetes cluster.
  - creating VLAN sub-interfaces.

    - creating L3 interfaces.

- Acts on pod events such as add and delete.

- Generates cRPD configuration.

The JCNR-CNI manages the secondary interfaces that the pods use. It creates the required interfaces based on the configuration in YAML-formatted network attachment definition (NAD) files. The JCNR-CNI configures some interfaces before passing them to their final location or connection point and provides an API for further interface configuration options such as:

- Instantiating different kinds of pod interfaces.

- Creating virtio-based high performance interfaces for pods that leverage the DPDK data plane.

- Creating veth pair interfaces that allow pods to communicate using the Linux Kernel networking stack.

- Creating pod interfaces in access or trunk mode.

- Attaching pod interfaces to bridge domains and virtual routers.

- Supporting IPAM plug-in for Dynamic IP address allocation.

- Allocating unique socket interfaces for virtio interfaces.

- Managing the networking tasks in pods such as assigning IP addresses and setting up of interfaces between the pod and vRouter in a Kubernetes cluster.

- Connecting pod interface to a network including pod-to-pod and pod-to-network.

- Integrating with the vRouter for offloading packet processing.

Benefits of JCNR-CNI:

- Improved pod interface management

- Customizable administrative and monitoring capabilities

- Increased performance through tight integration with the controller and vRouter components

The Role of JCNR-CNI in Pod Creation:

When you create a pod for use in the cloud-native router, the Kubernetes component known as **kubelet** calls the Multus CNI to set up pod networking and interfaces. Multus reads the annotations section of the **pod.yaml** file to find the NADs. If a NAD points to JCNR-CNI as the CNI plug in, Multus calls the JCNR-CNI to set up the pod interface. JCNR-CNI creates the interface as specified in the NAD. JCNR-CNI then generates and pushes a configuration into the controller.

Syslog-NG

Juniper Cloud-Native Router uses a syslog-ng pod to gather event logs from cRPD and vRouter and transform the logs into JSON-based notifications. The notifications are logged to a file. Syslog-ng runs as a daemonset.

Juniper Cloud-Native Router vRouter Datapath

SUMMARY

Cloud-Native Router supports both Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based vRouter forwarding plane.

IN THIS SECTION

The Cloud-Native Router vRouter forwarding plane supports both the Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath for high-speed packet processing.

Data Plane Development Kit (DPDK)

DPDK is an open-source set of libraries and drivers for rapid packet processing. DPDK enables fast packet processing by allowing network interface cards (NICs) to send direct memory access (DMA) packets directly into an application’s address space. This method of packet routing lets the application poll for packets, which prevents the overhead of interrupts from the NIC.

DPDK's poll mode drivers (PMDs) use the physical interface (NIC) of a VM's host instead of the Linux kernel's interrupt-based drivers. The NIC's registers operate in user space, which makes them accessible by DPDK’s PMDs. As a result, the host OS does not need to manage the NIC's registers. This means that the DPDK application manages all packet polling, packet processing, and packet forwarding of a NIC. Instead of waiting for an I/O interrupt to occur, a DPDK application constantly polls for packets and processes these packets immediately upon receiving them.

The vRouter dataplane is based off of DPDK 24.11.

eBPF XDP

> **NOTE**: This is a "Juniper Technology Preview (Tech Preview)" on page 333 feature.

Cloud-Native Router also supports an eBPF XDP datapath based vRouter. eBPF (extended Berkley Packet Filter) is a Linux kernel technology that executes user-defined programs inside a sandbox virtual machine. It enables low-level networking programs to execute with optimal performance. The eXpress Data Path (XDP) frameworks enables high-speed packet processing for the eBPF programs. Cloud-Native Router supports XDP in native (driver) mode on Baremental server deployments for limited drivers only. Please see the *System Requirements* for more details.

Benefits of eBPF XDP Datapath

Benefits of eBPF XDP Datapath include:

- An eBPF XDP kernel program and its custom library is easier to maintain across kernel versions and has wider kernel compatibility. The kernel dependencies are limited to a small set of eBPF helper functions.
- The program is safer since it is analysed by the in-built Linux eBPF verifier before it is loaded into the kernel.
- Offers higher performance using kernel bypass and omitting socket buffer (skb) allocation.

Supported Cloud-Native Router Features for eBPF XDP

The following Cloud-Native Router Features are supported with eBPF XDP for IPv4 traffic only:

- L3 traffic with Cloud-Native Router deployed as a sending, receiving or transit router
- VRF-Lite
- MPLSoUDP
- IGPs—OSPF, IS-IS
- BGP route advertisements

> **NOTE**: When deploying JCNR, you can configure the `agentModeType` attribute in the helmchart to select either a DPDK based or eBPF XDP datapath based vRouter.

Cloud-Native Router Deployment Modes

SUMMARY

Read this topic to know about the various modes of deploying the cloud-native router.

IN THIS SECTION

Deployment Modes

Starting with Juniper Cloud-Native Router Release 23.2, you can deploy and operate Juniper Cloud-Native Router in L2, L3 and L2-L3 modes, auto-derived based on the interface configuration in the `values.yaml` file prior to deployment.

NOTE: In the `values.yaml` file:

- When all the interfaces have an `interface_mode` key configured, then the mode of deployment would be L2.
- When one or more interfaces have an `interface_mode` key configured and some of the interfaces do not have the `interface_mode` key configured, then the mode of deployment would be L2-L3.
- When none of the interfaces have the `interface_mode` key configured, then the mode of deployment would be L3.

In L2 mode, the cloud-native router behaves like a switch and therefore does not performs any routing functions and it doesn not run any routing protocols. The pod network uses VLANs to direct traffic to various destinations.

In L3 mode, the cloud-native router behaves like a router and therefore performs routing functions and runs routing protocols such as ISIS, BGP, OSPF, and segment routing-MPLS. In L3 mode, the pod network is divided into an IPv4 or IPv6 underlay network and an IPv4 or IPv6 overlay network. The underlay network is used for control plane traffic.

The L2-L3 mode provides the functionality of both the switch and the router at the same time. It enables Cloud-Native Router to act as both a switch and a router simultaneously by performing switching in a set of interfaces and routing in the other set of interfaces. Cell site routers in a 5G deployment need to handle both L2 and L3 traffic. DHCP packets from radio outdoor unit (RU) is an example of L2 traffic and data packets moving from outdoor unit (ODU) to central unit (CU) is an example of L3 traffic.

Cloud-Native Router Interfaces Overview

SUMMARY

This topic provides information on the network communication interfaces provided by the JCNR-Controller. Fabric interfaces are aggregated interfaces that receive traffic from multiple interfaces. Interfaces to which different workloads are connected are called workload interfaces.

IN THIS SECTION

Read this topic to understand the network communication interfaces provided by the JCNR-Controller. We cover interface names, what they connect to, how they communicate and the services they provide.

Juniper Cloud-Native Router Interface Types

Juniper Cloud-Native Router supports two types of interfaces:

- **Fabric interfaces**—Aggregated interfaces that receive traffic from multiple interfaces. Fabric interfaces are always physical interfaces. They can either be a physical function (PF) or a virtual function (VF). The throughput requirement for these interfaces is higher, hence multiple hardware queues are allocated to them. Each hardware queue is allocated with a dedicated CPU core . The interfaces are configured for the cloud-native router using the appropriate `values.yaml` file in the deployer helmcharts. You can view the interface mapping using the `dpdkinfo -c` command (View the "Troubleshoot using the vRouter CLI" on page 315 topic for more details). You also have fabric workload interfaces that have low throughput requirement. Only one hardware queue is allocated to the interface, thereby saving precious CPU resources. These interfaces can be configured using the appropriate `values.yaml` file in the deployer helmcharts.

- **Workload interfaces**—Interfaces to which different workloads are connected. They can either be software-based or hardware-based interfaces. Software-based interfaces (pod interfaces) are either high-performance interfaces using the Data Plane Development Kit (DPDK) poll mode driver (PMD) or a low-performance interfaces using the kernel driver. Typically the DPDK interfaces are used for data traffic such as the GPRS Tunneling Protocol for user data (GTP-U) traffic and the kernel-based interfaces are used for control plane data traffic such as TCP. The kernel pod interfaces are typically for the operations, administration and maintenance (OAM) traffic or are used by non-DPDK pods. The kernel pod interfaces are configured as a veth-pair, with one end of the interface in the pod and the other end in the Linux kernel on the host. The DPDK native pod interfaces (virtio interfaces) are plumbed as vhost-user interfaces to the DPDK vRouter by the CNI. Cloud-Native Router also supports bonded interfaces via the link bonding PMD. These interfaces can be configured using the appropriate `values.yaml` file in the deployer helmcharts.

Cloud-Native Router supports different types of VLAN interfaces including trunk, access and sub-interfaces across fabric and workload interfaces.

Cloud-Native Router Interface Details

The different Cloud-Native Router interfaces are provided in detail below:

Agent Interface

The vRouter has only one agent interface. The agent interface enables communication between the vRouter-agent and the vRouter containers. On the vRouter CLI when you issue the `vif --list` command, the agent interface looks like this:

```
vif0/0      Socket: unix
            Type:Agent HWaddr:00:00:5e:00:01:00
            Vrf:65535 Flags:L2 QOS:-1 Ref:3
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:650  bytes:99307 errors:0
            Drops:0
```

L3 Fabric Interface (DPDK)

A layer-3 fabric interface bound to the DPDK.

L3 fabric interface in cRPD can be reviewed on the cRPD shell using the junos `show interfaces` command:

```
show interfaces routing ens2f2
Interface        State Addresses
ens2f2           Up    MPLS  enabled
                       ISO   enabled
```

```
                        INET  192.21.2.4
                        INET6 2001:192:21:2::4
                        INET6 fe80::c5da:7e9c:e168:56d7
                        INET6 fe80::a0be:69ff:fe59:8b58
```

The corresponding physical and tap interfaces can be seen on the vRouter using the `vif --list` command on the vRouter shell.

```
vif0/1      PCI: 0000:17:01.1 (Speed 25000, Duplex 1) NH: 7 MTU: 9000 <- PCI
Address
            Type:Physical HWaddr:d6:93:87:91:45:6c IPaddr: 192.21.2.4 <- Physical interface
            IP6addr:2001:192:21:2::4 <- IPv6 address
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:L3L2Vof QOS:0 Ref:16 <- L3 (only) interface
            RX port   packets:423168341 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:01.1  Status: UP  Driver: net_iavf
            RX packets:423168341  bytes:29123418594 errors:0
            TX packets:417508247  bytes:417226216530 errors:0
            Drops:8
            TX port   packets:417508247 errors:0
```

```
vif0/2      PMD: ens2f2 NH: 12 MTU: 9000 <- Tap interface name as seen by cRPD
            Type:Host HWaddr:d6:93:87:91:45:6c IPaddr: 192.21.2.4 <- Tap interface type
            IP6addr:2001:192:21:2::4
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:1  <-cross-connected to
vif 1
            RX device packets:306995  bytes:25719830 errors:0
            RX queue  packets:306995 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:306995  bytes:25719830 errors:0
            TX packets:307489  bytes:25880250 errors:0
            Drops:0
            TX queue  packets:307489 errors:0
            TX device packets:307489  bytes:25880250 errors:0
```

L3 Bond Interface (DPDK)

A layer 3 bond interface bound to DPDK.

```
show interfaces routing bond34
Interface        State Addresses
bond34           Up    INET6 2001:192:7:7::4
                       ISO   enabled
                       INET  192.7.7.4
                       INET6 fe80::527c:6fff:fe48:7574
```

```
vif0/3      PCI: 0000:00:00.0 (Speed 25000, Duplex 1) NH: 6 MTU: 1514 <- Bond interface (PCI id
0)
            Type:Physical HWaddr:50:7c:6f:48:75:74 IPaddr:192.7.7.4 <- Physical interface
            IP6addr:2001:192:7:7::4
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:TcL3L2Vof QOS:0 Ref:18
            RX port   packets:402183888 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond34  Status: UP  Driver: net_bonding <- Bonded master
            Slave Interface(0): 0000:5e:00.0  Status: UP  Driver: net_ice <- Bond slave - 1
            Slave Interface(1): 0000:af:00.0  Status: UP  Driver: net_ice <- Bond slave - 2
            RX packets:402183888  bytes:49519387070 errors:0
            TX packets:79226  bytes:7330912 errors:0
            Drops:1393
            TX port   packets:79226 errors:0
```

```
vif0/4      PMD: bond34 NH: 11 MTU: 9000
            Type:Host HWaddr:50:7c:6f:48:75:74 IPaddr:192.7.7.4 <- Tap interface
            IP6addr:2001:192:7:7::4
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:3 <- Tap interface for
bond
            RX device packets:76357  bytes:7101918 errors:0
            RX queue  packets:76357 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:76357  bytes:7101918 errors:0
            TX packets:75349  bytes:6946908 errors:0
            Drops:0
```

```
            TX queue  packets:75349 errors:0
            TX device packets:75349  bytes:6946908 errors:0
```

L3 Pod VLAN Sub-Interface (DPDK)

Starting in Juniper Cloud-Native Router Release 23.2, the cloud-native router supports the use of VLAN sub-interfaces in L3 mode, bound to DPDK.

Corresponding interface state in cRPD:

```
show interfaces routing ens1f0v1.201
Interface          State Addresses
ens1f0v1.201       Up    MPLS  enabled
                   ISO   enabled
                   INET6 fe80::b89c:fff:feab:e2c9
```

```
vif0/2      PCI: 0000:17:01.1 (Speed 25000, Duplex 1) NH: 7 MTU: 9000
            Type:Physical HWaddr:d6:93:87:91:45:6c IPaddr:0.0.0.0
            IP6addr:fe80::d493:87ff:fe91:456c <- IPv6 address
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:L3L2Vof QOS:0 Ref:16 <- L3 (only) interface
            RX port   packets:423168341 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:01.1  Status: UP  Driver: net_iavf
            RX packets:423168341  bytes:29123418594 errors:0
            TX packets:417508247  bytes:417226216530 errors:0
            Drops:8
            TX port   packets:417508247 errors:0
```

```
vif0/5      PMD: ens1f0v1 NH: 12 MTU: 9000
            Type:Host HWaddr:d6:93:87:91:45:6c IPaddr:0.0.0.0
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:2 <- L3 (only) tap
interface
            RX device packets:306995  bytes:25719830 errors:0
            RX queue  packets:306995 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:306995  bytes:25719830 errors:0
```

```
            TX packets:307489  bytes:25880250
errors:0

            Drops:0
            TX queue  packets:307489 errors:0
            TX device packets:307489  bytes:25880250 errors:0
```

```
vif0/9      Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/2 NH: 36 MTU: 1514 <- VLAN
fabric sub-intf with parent as vif 2 and VLAN tag as 201
            Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:L3DProxyEr QOS:-1 Ref:4
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

```
vif0/10     Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/5 NH: 21 MTU: 9000
            Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:4 TxXVif:9 <- VLAN tap sub-intf
cross connected to fabric sub-intf vif 9 and parent as tap intf vif 5
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

```
vif0/50     PMD: vhostnet1-9403fd77-648a-47 NH: 177 MTU: 9160                     ---> pod
interface
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:65535 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:20
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
```

```
            TX packets:0  bytes:0 errors:0
            Drops:0
```

```
vif0/51     Virtual: vhostnet1-9403fd77-648a-47.201 Vlan(o/i)(,S): 201/201 NH: 17 MTU: 1514
            Parent:vif0/50                                                          ---->L3 pod
sub-interface, parent is the pod interface
            Type:Virtual(Vlan) HWaddr:00:00:5e:00:01:00 IPaddr:99.62.0.2
            IP6addr:1234::633e:2
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:PL3DProxyEr QOS:-1 Ref:4
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

L3 Pod Kernel Interface

These are non-DPDK L3 pod interfaces. Interface state in the cRPD:

```
show interfaces routing jvknet1-0af476e
Interface        State Addresses
jvknet1-0af476e  Up    INET6 enabled
                       INET6 abcd:2:51:1::4
                       ISO   enabled
                       INET  enabled
                       INET  2.51.1.4
```

```
vif0/13     Ethernet: jvknet1-0af476e NH: 35 MTU: 9160 <- Kernel interface (jvk) of CNF
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:2.51.1.4 <- pod/ workload
            IP6addr:abcd:2:51:1::4
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:PL3DVofProxyEr QOS:-1 Ref:11
            RX port   packets:47 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:47  bytes:13012 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:47
```

L2 Fabric Interface (DPDK, Physical Trunk)

DPDK L2 fabric interfaces, which are associated with the physical network interface card (NIC) on the host server, accept traffic from multiple VLANs. The trunk interfaces accept only tagged packets. Any untagged packets are dropped. These interfaces can accept a VLAN filter to allow only specific VLAN packets. A trunk interface can be a part of multiple bridge-domains (BD). A bridge domain is a set of logical ports that share the same flooding or broadcast characteristics. Like a VLAN, a bridge domain spans one or more ports of multiple devices.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    ens786f0v0 {
        unit 0 {
            family bridge {
                interface-mode trunk;
                vlan-id-list 1001-1100;
            }
        }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the DPDK VF fabric interface looks like this:

```
vif0/1    PCI: 0000:31:01.0 (Speed 10000, Duplex 1)
          Type:Physical HWaddr:d6:22:c5:42:de:c3
          Vrf:65535 Flags:L2Vof QOS:-1 Ref:12
          RX queue packets:11813 errors:1
          RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 1 0
          Fabric Interface: 0000:31:01.0 Status: UP Driver: net_iavf
          Vlan Mode: Trunk Vlan: 1001-1100
          RX packets:0 bytes:0 errors:49962
          TX packets:18188356 bytes:2037400554 errors:0
          Drops:49963
```

DPDK L2 Bond Interface (Active-Standby, Trunk)

Layer-2 Bond interfaces accept traffic from multiple VLANs. A bond interface runs in the active or standby mode (mode 0). You define the bond interface in the helm chart configuration as follows:

```
bondInterfaceConfigs:
- name: "bond0"
  mode: 1          # ACTIVE_BACKUP MODE
  slaveInterfaces:
  - "ens2f0v1"
  - "ens2f1v1"
```

```
 - bond0:
      ddp: "auto"
      interface_mode: trunk
      vlan-id-list: [1001-1100]
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 1001
      no-local-switching: true
```

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    bond0 {
         unit 0 {
             family bridge
             interface-mode trunk;
             vlan-id-list 1001-1100;
         }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the bond interface looks like this:

```
vif0/2      PCI: 0000:00:00.0 (Speed 10000, Duplex 1)
            Type:Physical HWaddr:32:f8:ad:8c:d3:bc
            Vrf:65535 Flags:L2Vof QOS:-1 Ref:8
            RX queue  packets:1882 errors:0
```

```
RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
Fabric Interface: eth_bond_bond0  Status: UP  Driver: net_bonding
Slave Interface(0): 0000:81:01.0  Status: UP  Driver: net_iavf
Slave Interface(1): 0000:81:03.0  Status: UP  Driver: net_iavf
Vlan Mode: Trunk  Vlan: 1001-1100
RX packets:8108366000  bytes:486501960000 errors:4234
TX packets:65083776  bytes:4949969408 errors:0
Drops:8108370394
```

DPDK L2 Pod Interface (Virtio Trunk)

The trunk interfaces accept only tagged packets. Any untagged packets are dropped. These interfaces can accept a VLAN filter to allow only specific VLAN packets. A trunk interface can be a part of multiple bridge-domains (BD). A bridge domain is a set of logical ports that share the same flooding or broadcast characteristics. Like a VLAN, a bridge domain spans one or more ports of multiple devices. Virtio interfaces are associated with pod interfaces that use virtio on the DPDK data plane.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    vhost242ip-93883f16-9ebb-4acf-b {
        unit 0 {
            family bridge {
                interface-mode trunk;
                vlan-id-list 1001-1003;
            }
        }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the virtio with DPDK data plane interface looks like this:

```
vif0/3    PMD: vhost242ip-93883f16-9ebb-4acf-b
          Type:Virtual HWaddr:00:16:3e:7e:84:a3
          Vrf:65535 Flags:L2 QOS:-1 Ref:13
          RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          Vlan Mode: Trunk Vlan: 1001-1003
          RX packets:0 bytes:0 errors:0
          TX packets:10604432 bytes:1314930908 errors:0
```

```
          Drops:0
          TX port packets:0 errors:10604432
```

L2 Pod Kernel Interface (Access)

The access interfaces accept both tagged and untagged packets. Untagged packets are tagged with the access VLAN or access BD. Any tagged packets other than the ones with access VLAN are dropped. The access interfaces is a part of a single bridge-domain. It does not have any parent interface.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
routing-instances {
    switch {
         instance-type virtual-switch;
         bridge-domains
{

             bd1001 {
                 vlan-id 1001;
                 interface jvknet1-eed79ff;
             }
         }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the veth pair interface looks like this:

```
vif0/4      Ethernet: jvknet1-88c44c3
            Type:Virtual HWaddr:02:00:00:3a:8f:73
            Vrf:0 Flags:L2Vof QOS:-1 Ref:10
            RX queue packets:524 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Vlan Mode: Access Vlan Id: 1001 OVlan Id: 1001
            RX packets:9 bytes:802 errors:515
            TX packets:0 bytes:0 errors:0
            Drops: 525
```

L2 Pod VLAN Sub-interface (DPDK)

You can configure a user pod with a Layer 2 VLAN sub-interface and attach it to the Cloud-Native Router instance. VLAN sub-interfaces are like logical interfaces on a physical switch or router. They access only tagged packets that match the configured VLAN tag. A sub-interface has a parent interface. A parent interface can have multiple sub-interfaces, each with a VLAN ID. When you run the cloud-native router, you must associate each sub-interface with a specific VLAN.

The cRPD interface configuration viewed using the `show configuration` command is as shown below (the output is trimmed for brevity).

For **L2**:

```
routing-instances {
    switch {
         instance-type virtual-switch;
         bridge-domains
{

             bd3003 {
                 vlan-id 3003;
                 interface vhostnet1-71cd7db1-1a5e-49.3003;
             }
         }
     }
}
```

On the vRouter, a VLAN sub-interface configuration is as shown below:

```
vif0/4      PMD: vhostnet1-71cd7db1-1a5e-49 MTU: 9160
            Type:Virtual HWaddr:02:00:00:84:dc:42
            DDP: OFF SwLB: ON
            Vrf:65535 Flags:L2 QOS:-1 Ref:14
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
            TX port   packets:0 errors:293

vif0/5      Virtual: vhostnet1-71cd7db1-1a5e-49.3003 Vlan(o/i)(,S): 3003/3003 Parent:vif0/4
            Type:Virtual(Vlan) HWaddr:00:99:99:99:33:09
            Vrf:0 Flags:L2 QOS:-1 Ref:3
```

```
        RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
        RX packets:0  bytes:0 errors:0
        TX packets:0  bytes:0 errors:0
        Drops:0
```

RELATED DOCUMENTATION

Cloud-Native Router Use-Cases and Configuration Overview | 224

2

CHAPTER

L2 Features

IN THIS CHAPTER

L2 Features Overview

SUMMARY

Read this topic to learn about the features available in the Juniper Cloud-Native Router when deployed in L2 (switch) mode.

The Juniper Cloud-Native Router supports multiple "deployment modes" on page 13.

In L2 mode, the cloud-native router behaves like a switch and so performs no routing functions and runs no routing protocols. The pod network uses VLANs to direct traffic to various destinations.

This chapter provides information about the various L2 features supported by JCNR.

Layer 2 Circuit

SUMMARY

Juniper-Cloud Native Router supports Layer 2 circuits over an IP/MPLS-based service provider's network. This topic provides configuration and verification details.

IN THIS SECTION

Juniper Cloud-Native Router supports Layer 2 circuits for a point-to-point Layer 2 connection over an IP/MPLS-based service provider's network. To establish the Layer 2 circuit, it uses Label Distribution Protocol (LDP) as the signaling protocol to advertise the ingress label to the remote PE routers. A targeted LDP session is established between the loopback addresses of the two PEs to exchange VPN labels. For more information on L2 circuits, please review Layer 2 Circuit Overview.

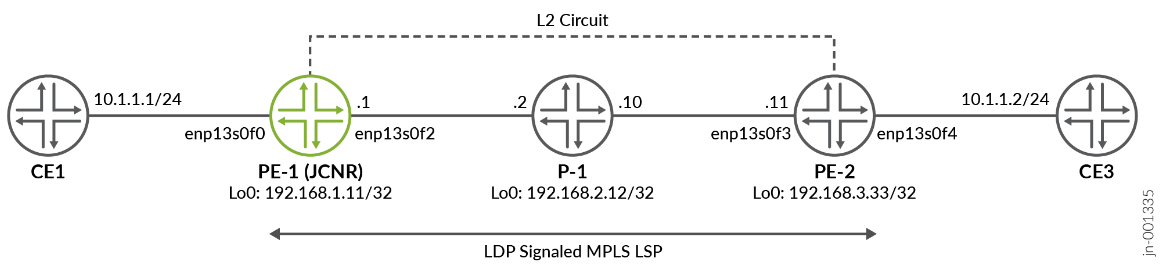
The following Layer-2 circuit features are supported by the Cloud-Native Router:

- Enable `l2circuit` protocol on Physical (PF/VF), VLAN sub-interface, bond interface (towards core), and pod interfaces
- Ethernet CCC encapsulations— `ethernet-ccc` and `vlan-ccc`

- Local interface switching
- Control word
- Protect Interface for CE redundancy
- Backup Neighbor for Core redundancy
- Pseudowire cold and hot-standby
- Support L2 circuit with BGP-Labeled Unicast (BGP-LU)
- Support for dual VLAN tagging
- Interoperability with other Junos devices

Configuration

You can configure Layer 2 circuits in Juniper Cloud-Native Router using *configlets*. Multiple configlet samples are provided in this section based on the following topology:



For Layer 2 circuit configuration, you configure the Ethernet-based CE-facing interface with the CCC encapsulation type of your choice—`ethernet-ccc` or `vlan-ccc`. The Layer 2 circuit configuration such as the remote PE neighbour (usually the loopback address), the interface connected to the CE-router, and a virtual circuit identifier for the virtual circuit (VC) is performed under the `edit protocols l2circuit` statement. Eventually, you configure MPLS, LDP and an IGP to enable signaling for your Layer 2 circuit. Please review Example: Enternet-based Layer 2 Circuit Configuration for a end-to-end Junos configuration example.

Layer 2 Circuit with ethernet-ccc

Configure Ethernet CCC encapsulation on CE-facing Ethernet interfaces on PE-1 (JCNR) that must accept packets carrying standard Tag Protocol ID (TPID) values.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 0 encapsulation ethernet-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0 virtual-circuit-id 100
    set protocols ldp interface enp13s0f2
    set protocols ldp interface lo0.0
    set protocols mpls interface enp13s0f2.0
    set protocols ospf area 0.0.0.0 interface lo0.0
    set protocols ospf area 0.0.0.0 interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Layer 2 Circuits with vlan-ccc

Configure VLAN CCC encapsulation on CE-facing Ethernet interfaces on PE-1 (JCNR) with VLAN tagging enabled.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 102 vlan-id 102
    set interfaces enp13s0f0 unit 102 encapsulation vlan-ccc
```

```
    set interfaces enp13s0f0 unit 104 vlan-id 104
    set interfaces enp13s0f0 unit 104 encapsulation vlan-ccc
    set interfaces enp13s0f0 unit 106 vlan-id 106
    set interfaces enp13s0f0 unit 106 encapsulation vlan-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.102 virtual-circuit-id 102
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.104 virtual-circuit-id 104
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.106 virtual-circuit-id 106
    set protocols ldp interface enp13s0f2
    set protocols ldp interface lo0.0
    set protocols mpls interface enp13s0f2.0
    set protocols ospf area 0.0.0.0 interface lo0.0
    set protocols ospf area 0.0.0.0 interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Layer 2 Circuit with Local Switching

You can configure Layer 2 circuit with local switching. Optionally, configure `protect-interface` for local or remote end to ensure traffic switch over when the primary interface goes down. Protect interfaces act as backups for their associated interfaces that link a virtual circuit to its destination. The primary interface has priority over the protect interface and carries network traffic as long as it is functional. If the primary interface fails, the protect interface is activated. Optionally, configure `no-revert` to prevent switch back to primary when primary interface is back up.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 0 encapsulation ethernet-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0 virtual-circuit-id 100
    set protocols l2circuit local-switching interface enp13s0f0 no-revert protect-interface
```

```
enp13s0f1 end-interface interface enp13s0f3 no-revert protect-interface enp13s0f8
    set protocols ldp interface enp13s0f2
    set protocols ldp interface lo0.0
    set protocols mpls interface enp13s0f2.0
    set protocols ospf area 0.0.0.0 interface lo0.0
    set protocols ospf area 0.0.0.0 interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Layer 2 Circuit with Ignore Mismatch

You can configure the Layer 2 circuit to establish even though the MTU (`ignore-mtu-mismatch`), encapsulation (`ignore-encapsulation-mismatch`) or VLAN ID (`no-vlan-id-validate`) configured on the CE device interface does not match the setting configured on the Layer 2 circuit interface.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 0 encapsulation ethernet-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0 virtual-circuit-id 100
ignore-encapsulation-mismatch ignore-mtu-mismatch no-vlan-id-validate
    set protocols ldp interface enp13s0f2
    set protocols ldp interface lo0.0
    set protocols mpls interface enp13s0f2.0
    set protocols ospf area 0.0.0.0 interface lo0.0
    set protocols ospf area 0.0.0.0 interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Static Layer 2 Circuits

Configure static Layer 2 circuit pseudowires for networks that do not support LDP or do not have LDP enabled. Static pseudowires require you to configure static values for the in and out labels that enable a pseudowire connection.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 102 vlan-id 102
    set interfaces enp13s0f0 unit 102 encapsulation vlan-ccc
    set interfaces enp13s0f0 unit 104 vlan-id 104
    set interfaces enp13s0f0 unit 104 encapsulation vlan-ccc
    set interfaces enp13s0f0 unit 106 vlan-id 106
    set interfaces enp13s0f0 unit 106 encapsulation vlan-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.102 static incoming-label
103
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.102 static outgoing-label
102
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.102 virtual-circuit-id 102
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.104 static incoming-label
105
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.104 static outgoing-label
104
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.104 virtual-circuit-id 104
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.106 static incoming-label
107
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.106 static outgoing-label
106
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.106 virtual-circuit-id 106
    set protocols ldp interface enp13s0f2
    set protocols mpls label-range static-label-range 16 200
    set protocols mpls interface enp13s0f2.0
    set protocols ospf area 0.0.0.0 interface lo0.0
```

```
    set protocols ospf area 0.0.0.0 interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Layer 2 Circuit with BGP-LU

Enable Layer 2 circuit over BGP-Labeled Unicast (BGP-LU). BGP-LU advertises the ingress label to its peer PE routers.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-l2-ckt-pe1
  namespace: jcnr
spec:
  config: |-
    set interfaces enp13s0f2 unit 0 family inet address 172.16.25.1/24
    set interfaces enp13s0f0 description "to CE-1 7/4"
    set interfaces enp13s0f0 unit 102 vlan-id 102
    set interfaces enp13s0f0 unit 102 encapsulation vlan-ccc
    set interfaces enp13s0f0 unit 104 vlan-id 104
    set interfaces enp13s0f0 unit 104 encapsulation vlan-ccc
    set interfaces enp13s0f0 unit 106 vlan-id 106
    set interfaces enp13s0f0 unit 106 encapsulation vlan-ccc
    set interfaces lo0 unit 0 family inet address 192.168.1.11/32
    set policy-options policy-statement local-prefixes from protocol direct
    set policy-options policy-statement local-prefixes from prefix-list local-prefixes
    set policy-options policy-statement local-prefixes then accept
    set policy-options policy-statement send-pe from route-filter 192.168.1.11/32 exact
    set policy-options policy-statement send-pe then accept
    set policy-options prefix-list local-prefixes 192.168.1.11/32
    set routing-options router-id 192.168.1.11
    set routing-options autonomous-system 65001
    set routing-options rib-groups INET0_to_INET3 import-rib inet.0
    set routing-options rib-groups INET0_to_INET3 import-rib inet.3
    set protocols bgp group external type external
    set protocols bgp group external local-address 172.16.25.1
    set protocols bgp group external family inet labeled-unicast rib inet.3
    set protocols bgp group external family inet unicast rib-group INET0_to_INET3
    set protocols bgp group external export local-prefixes
```

```
    set protocols bgp group external neighbor 172.16.30.11 peer-as 65002
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.102 virtual-circuit-id 102
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.104 virtual-circuit-id 104
    set protocols l2circuit neighbor 192.168.3.33 interface enp13s0f0.106 virtual-circuit-id 106
    set protocols ldp interface lo0.0
    set protocols mpls interface enp13s0f2
  crpdSelector:
    matchLabels:
      kubernetes.io/hostname: node-1
```

Configuring via JCNR-CNI (Pod Configuration)

When using the Cloud-Native Router in CNI mode, you can configure cRPD with layer 2 circuit configuration using the JCNR-CNI. Here is an example pod configuration:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: jcnr-l2vpn-tests
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: l2ckt-pod1-nad
  namespace: jcnr-l2vpn-tests
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "l2ckt-pod1-nad",
    "plugins": [
      {
        "type": "jcnr",
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'

---
apiVersion: v1
kind: Pod
```

```
metadata:
  name: l2ckt-pod1-nad
  namespace: jcnr-l2vpn-tests
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "l2ckt-pod1-nad",
          "interface": "net1",
          "cni-args": {
            "l2CktNbr": "192.168.3.33",
            "l2CktVcid": "10"
          }
        }
      ]
spec:
  containers:
... <trimmed>
```

Verification

You can verify the Layer 2 circuit configuration and statistics on the "cRPD shell" on page 329 and "vRouter shell" on page 331.

Verify L2 circuit Connections

Display status information about Layer 2 virtual circuits from the Cloud-Native Router to its neighbors using `show l2circuit connections` command on the cRPD.

```
user@host> show l2circuit connections
Layer-2 Circuit Connections:

Legend for connection status (St)
EI -- encapsulation invalid      NP -- interface h/w not present
MM -- mtu mismatch               Dn -- down
EM -- encapsulation mismatch     VC-Dn -- Virtual circuit Down
CM -- control-word mismatch      Up -- operational
VM -- vlan id mismatch        CF -- Call admission control failure
OL -- no outgoing label          IB -- TDM incompatible bitrate
```

```
NC -- intf encaps not CCC/TCC    TM -- TDM misconfiguration
BK -- Backup Connection          ST -- Standby Connection
CB -- rcvd cell-bundle size bad  SP -- Static Pseudowire
LD -- local site signaled down   RS -- remote site standby
RD -- remote site signaled down  HS -- Hot-standby Connection
XX -- unknown

Legend for interface status
Up -- operational
Dn -- down
Neighbor: 192.168.3.33
    Interface                 Type  St     Time last up          # Up trans
    enp13s0f0.102(vc 102)     rmt   Up     Mar 12 15:45:20 2025           2
      Remote PE: 192.168.3.33, Negotiated control-word: Yes (Null)
      Incoming label: 17, Outgoing label: 16
      Negotiated PW status TLV: No
      Local interface: enp13s0f0.102, Status: Up, Encapsulation: VLAN
      Flow Label Transmit: No, Flow Label Receive: No
    enp13s0f0.104(vc 104)     rmt   Up     Mar 12 15:45:24 2025           1
      Remote PE: 192.168.3.33, Negotiated control-word: Yes (Null)
      Incoming label: 18, Outgoing label: 17
      Negotiated PW status TLV: No
      Local interface: enp13s0f0.104, Status: Up, Encapsulation: VLAN
      Flow Label Transmit: No, Flow Label Receive: No
    enp13s0f0.106(vc 106)     rmt   Up     Mar 12 15:45:24 2025           1
      Remote PE: 192.168.3.33, Negotiated control-word: Yes (Null)
      Incoming label: 19, Outgoing label: 18
      Negotiated PW status TLV: No
      Local interface: enp13s0f0.106, Status: Up, Encapsulation: VLAN
      Flow Label Transmit: No, Flow Label Receive: No
```

Verify the LDP sessions

Display information about LDP sessions using the `show ldp session` command on the cRPD.

```
user@root> show ldp session extensive
Address: 192.168.3.33, State: Operational, Connection: Open, Hold time: 23
  Session ID: 192.168.1.11:0--192.168.3.33:0
  Next keepalive in 3 seconds
  Passive, Maximum PDU: 4096, Hold time: 30, Neighbor count: 1
  Neighbor types: configured-layer2
```

```
  Keepalive interval: 10, Connect retry interval: 1
  Local address: 192.168.1.11, Remote address: 192.168.3.33
  Up for 01:13:18
  Capabilities advertised: none
  Capabilities received: none
  Protection: disabled
  Session flags: none
  Local - Restart: disabled, Helper mode: enabled
  Remote - Restart: disabled, Helper mode: enabled
  Local maximum neighbor reconnect time: 120000 msec
  Local maximum neighbor recovery time: 240000 msec
  Local Label Advertisement mode: Downstream unsolicited
  Remote Label Advertisement mode: Downstream unsolicited
  Negotiated Label Advertisement mode: Downstream unsolicited
  MTU discovery: disabled
  Nonstop routing state: Not in sync
  Next-hop addresses received:
    192.168.3.33
  Queue depth: 0
Message type          Total            Last 5 seconds
               Sent     Received       Sent         Received
Initialization          1             1             0                0
Keepalive             439           439             1                1
Notification            0             0             0                0
Address                 1             1             0                0
Address withdraw        0             0             0                0
Label mapping           7             7             0                0
Label request           0             0             0                0
Label withdraw          3             3             0                0
Label release           3             3             0                0
Label abort             0             0             0                0
```

Verify LDP Database

```
user@root> show ldp database
Input label database, 192.168.1.11:0--192.168.3.33:0
Labels received: 14
  Label     Prefix
     27      L2CKT CtrlWord ETHERNET VC 9
     28      L2CKT CtrlWord ETHERNET VC 10
```

```
Output label database, 111.1.1.1:0--133.3.3.3:0
Labels advertised: 14
  Label     Prefix
     36      L2CKT CtrlWord ETHERNET VC 9
     37      L2CKT CtrlWord ETHERNET VC 10
```

Verify vRouter Interfaces

Verify the vRouter interfaces for CE-facing interfaces.

```
bash-5.1# vif --list
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled
       LsDp=Link down in DP only, Ccc=CCC Enabled, HwTs=Hardware support for Timestamp
...<trimmed>

vif0/2      PCI: 0000:0d:00.0 (Speed 10000, Duplex 1) NH: 7 MTU: 9000
            Type:Physical HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:TcL3Vof QOS:0 Ref:18
            RX port   packets:899840301 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:0d:00.0  Status: UP  Driver: net_ice
            RX packets:899840302  bytes:4080744017446 errors:0
            TX packets:896126245  bytes:4059828585689 errors:0
            Drops:1315423
            TX port   packets:896126245 errors:0

vif0/9      PMD: enp13s0f0 NH: 21 MTU: 9000
            Type:Host HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
```

```
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3ProxyEr QOS:-1 Ref:17 TxXVif:2
            RX device packets:70  bytes:6064 errors:0
            RX queue  packets:70 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:70  bytes:6064 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/16     Virtual: enp13s0f0.102 Vlan(o/i)(,S): 102/102
            Parent:vif0/9  Sub-type:  Host-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3ProxyEr QOS:-1 Ref:1 TxXVif:17
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:8  bytes:592 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/17     Virtual: enp13s0f0.102 Vlan(o/i)(,S): 102/102 NH: 44
            Parent:vif0/2  Sub-type:  physical-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:Ccc QOS:-1 Ref:5
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:299403723  bytes:1357784724696 errors:0
            TX packets:298673909  bytes:1351924185670 errors:0
            Drops:612067

vif0/18     Virtual: enp13s0f0.104 Vlan(o/i)(,S): 104/104
            Parent:vif0/9  Sub-type:  Host-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3ProxyEr QOS:-1 Ref:1 TxXVif:19
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:9  bytes:666 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/19     Virtual: enp13s0f0.104 Vlan(o/i)(,S): 104/104 NH: 48
            Parent:vif0/2  Sub-type:  physical-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
```

```
            Vrf:0 Mcast Vrf:0 Flags:Ccc QOS:-1 Ref:5
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:299381730  bytes:1357688891917 errors:0
            TX packets:298673939  bytes:1351924259113 errors:0
            Drops:593380

vif0/20     Virtual: enp13s0f0.106 Vlan(o/i)(,S): 106/106
            Parent:vif0/9  Sub-type:  Host-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3ProxyEr QOS:-1 Ref:1 TxXVif:21
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:8  bytes:592 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/21     Virtual: enp13s0f0.106 Vlan(o/i)(,S): 106/106 NH: 56
            Parent:vif0/2  Sub-type:  physical-tap
            Type:Virtual(Vlan) HWaddr:40:a6:b7:c4:23:f4 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:Ccc QOS:-1 Ref:5
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:299359992  bytes:1357592722594 errors:0
            TX packets:298673884  bytes:1351924089886 errors:0
            Drops:571666
```

Verify Interface Rate Statistics

Verify the interface statistics inclusing received and transmitted packets and errors.

```
bash-5.1# vif –list –rate

Interface rate statistics
-------------------------


Interface name                                                          VIF
ID                        RX                             TX


       Errors    Packets              Errors    Packets


Agent: unix
```

```
vif0/0                0      0                 0      0
Physical: eth_bond_bond0
vif0/1                0      397296            0      402976
Physical: 0000:0d:00.0
vif0/2                0      201471            0      197687
Physical: 0000:0d:00.1
vif0/3                0      0                 0      0
Physical: 0000:0d:00.2
vif0/4                0      200853            0      199410
Physical: 0000:0d:00.3
vif0/5                0      0                 0      0
Physical: 0000:b5:00.2
vif0/6                0      0                 0      0
Physical: 0000:b5:00.3
vif0/7                0      0                 0      0
Host: bond0
vif0/8                0      0                 0      0
Host: enp13s0f0
vif0/9                0      0                 0      0
Host: enp13s0f1
vif0/10               0      0                 0      0
Host: enp13s0f2
vif0/11               0      0                 0      0
Host: enp13s0f3
vif0/12               0      0                 0      0
Host: enp181s0f2
vif0/13               0      0                 0      0
Host: enp181s0f3
vif0/14               0      0                 0      0
Virtual(Vlan): enp13s0f0.102
vif0/16               0      0                 0      0

Key 'q' for quit, key 'k' for previous page, key 'j' for next page.
2025-03-12 16:58:44 +0000
```

Verify Routes and Nexthops

Verify the routes and nexthops for interfaces with Ethernet CCC encapsulation enabled.

```
bash-5.1# rt --dump 0 --family ccc
Flags: L=Label Valid, L2c=L2 Control Word
```

```
vRouter ccc table 0/0
Interface Id              Flags           Label/VNID      Nexthop           Stats
       21                LL2c                 18            61                0
       19                LL2c                 17            61                0
        4                LL2c                110            40                0
       17                LL2c                 16            61                0
```

```
bash-5.1# nh --get 61
Id:61         Type:Tunnel         Fmly: AF_MPLS  Rid:0  Ref_cnt:4         Vrf:0
              Flags:Valid, Etree Root, MPLS,
              Oif:1 Len:14 Data:36 d0 5b f4 f3 ef a2 2a 5f 77 e7 8b 88 47 Number of Transport
Labels:1 Transport Labels:31,
```

```
bash-5.1# nh --get 40
Id:40         Type:Tunnel         Fmly: AF_MPLS  Rid:0  Ref_cnt:2         Vrf:0
              Flags:Valid, Etree Root, MPLS,
              Oif:1 Len:14 Data:36 d0 5b f4 f3 ef a2 2a 5f 77 e7 8b 88 47 Number of Transport
Labels:1 Transport Labels:18,
```

Loop Detection in Pure L2 Mode

SUMMARY

Juniper Cloud-Native Router supports Layer 2 loop detection mechanisms by detecting frequent MAC address movements between ports.

Juniper Cloud-Native Router supports Layer 2 loop detection mechanism in the vRouter data path. Frequent MAC address movements between ports, many a times resulting from incorrect wiring, can result in L2 loops, resulting in network instability, broadcast storms, and degragded performance. Traditional loop detection mechanisms such as Spanning Tree Protocol (STP) may not always be feasible in modern data center environments.

Cloud-Native Router implements a loop detection mechanism that identifies MAC address learning loops by monitoring the frequency of MAC address movements between ports. The vRouter uses the MAC-MOVE table to track these MAC movemements, including source MAC, VLAN tag, hit count, and timestamp. The vRouter detects high hit counts on the same MAC address entry in the MAC_MOVE table, indicating continuous movement of the MAC address between two ports within a short interval of time.

You can use the **purel2cli --mac-move-table show** command on the "vRouter shell" on page 331 to view the MAC_MOVE table:

```
purel2cli --mac-move-table show
===================================================
||  MAC            vlan      hit_count     timestamp||
===================================================
00:01:01:01:01:03  1221      46            1731038878
Total Mac entries 1
```

The vRouter also logs the detected loop and shuts down affected ports.

```
2024-11-18 06:14:19,130 DPCORE: NOTIFICATION, CRIT, Purel2 Macmove Loop Detected, mac
00:01:01:01:01:03, vlan 1221, Current Port 5, Older Port 1, hitcount 52
2024-11-18 06:14:19,130 VROUTER: Func: dpdk_port_shutdown, Line 1296, Shutting Down Port 0 of
Vif 5
2024-11-18 06:14:19,343 lcore 17 called tx_pkt_burst for not ready port (
2024-11-18 06:14:23,242 VROUTER: Port ID: 4 Link Status: DOWN intf _name:0000:d8:00.0
drv_name:net_ixgbe

2024-11-18 06:14:23,242 VROUTER: Notifed Link status update to agent for interface 0000: d8:00.0
as 0
2024-11-18 06:14:36,299 DPCORE: vr_purel2_mac_move_table_req_process
vr_purel2_mac_move_table_req 456 received OP 1

2024-11-18 06 14:38,133 DPCORE: vr_purel2_mac_move_table_req_process
vr_purel2_mac_move_table_req 456 received OP 1
```

A Syslog notification is also generated when the loop is detected:


```
{"jcnr": {"header": {"sysUpTime": "40 days, 11 hours, 35 minutes, 45 seconds" 45 "program":
"vrouter-dpdk", "notificationType": "DPCORE: ", "eventDate": "2024-12-02T21:32:23-08:00"} ,
```

```
"body": "NOTIFICATION, CRIT, Purel2 Macmove Loop Detected, mac 00:01:01:01:01:03, vlan 1221.
Current Port 4, Older Port 2, hitcount 51"}}
```

Access Control Lists (Firewall Filters)

SUMMARY

Read this topic to learn about Layer 2 access control lists (Firewall filters) in the cloud-native router.

IN THIS SECTION

Access Control Lists (Firewall Filters)

Starting with Juniper Cloud-Native Router Release 22.2 we've included a limited firewall filter capability. You can configure the filters using the Junos OS CLI within the cloud-native router controller, using NETCONF, or the cloud-native router APIs. Starting with Juniper Cloud-Native Router Release 23.2, you can also configure firewall filters using node annotations and custom configuration template at the time of Cloud-Native Router deployment. Please review the deployment guide for more details.

During deployment, the system defines and applies firewall filters to block traffic from passing directly between the router interfaces. You can dynamically define and apply more filters. Use the firewall filters to:

- Define firewall filters for bridge family traffic.
- Define filters based on one or more of the following fields: source MAC address, destination MAC address, or EtherType.
- Define multiple terms within each filter.
- Discard the traffic that matches the filter.
- Apply filters to bridge domains.

Configuration Example

Below you can see an example of a firewall filter configuration from a cloud-native router deployment:

```
root@jcnr01> show configuration firewall
firewall {
    family {
        bridge {
            filter example {
                term t1 {
                    from {
                        destination-mac-address 10:10:10:10:10:11;
                        source-mac-address 10:10:10:10:10:10;
                        ether-type arp;
                    }
                    then {
                        discard;
                    }
                }
            }
        }
    }
}
```

NOTE: You can configure up to 16 terms in a single firewall filter. The only *then* action you can configure in a firewall filter is the `discard` action.

After configuration, you must apply your firewall filters to a bridge domain using the `set routing-instances vswitch bridge-domains` *`bd3001`* `forwarding-options filter input` *`filter1`* configuration command. Then you must commit the configuration for the firewall filter to take effect.

To see how many packets matched the filter (per VLAN), you can issue the `show firewall filter` *`filter1`* command on the controller CLI. For example:

```
show firewall filter filter1
 Filter : filter1    vlan-id : 3001
 Term                  Packet
  t1                   0
```

In the preceding example, we applied the filter to the bridge domain `bd3001`. The filter has not yet matched any packets.

Troubleshooting

The following table lists some of the potential problems that you might face when you implement firewall rules or ACLs in the cloud-native router. You run most of these commands on the host server.

Table 1: L2 Firewall Filter or ACL Troubleshooting

Problem	Possible Causes and Resolution	Command
Firewall filters or ACLs not working	gRPC connection (port 50052) to the vRouter is down. Check the gRPC connection.	`netstat -antp|grep 50052`
	The `ui-pubd` process is not running. Check whether `ui-pubd` is running.	`ps aux|grep ui-pubd`
Firewall filter or ACL show commands not working	The gRPC connection (port 50052) to the vRouter is down. Check the gRPC connection.	`netstat -antp|grep 50052`
	The firewall service is not running.	`ps aux|grep firewall`
		`show log filter.log` You must run this command in the JCNR-controller (cRPD) CLI.

MAC Learning and Aging

SUMMARY

Juniper Cloud-Native Router provides automated learning and aging of MAC addresses. Read this topic

IN THIS SECTION

for an overview of the MAC learning and aging functionality in the cloud-native router.	

MAC Learning

MAC learning enables the cloud-native router to efficiently send the received packets to their respective destinations. The cloud-native router maintains a table of MAC addresses grouped by interface. The table includes MAC addresses, VLANs, and the interface on which the vRouter learns each MAC address and VLAN. The MAC table informs the vRouter about the MAC addresses that each interface can reach.

The cloud-native router caches the source MAC address for a new packet flow to record the incoming interface into the MAC table. The router learns the MAC addresses for each VLAN or bridge domain. The cloud-native router creates a key in the MAC table from the MAC address and VLAN of the packet. Queries sent to the MAC table return the interface associated with the key. To enable MAC learning, the cloud-native router performs these steps:

- Records the incoming interface into the MAC table by caching the source MAC address for a new packet flow.
- Learns the MAC addresses for each VLAN or bridge domain.
- Creates a key in the MAC table from the MAC address and VLAN of the packet.

If the destination MAC address and VLAN are missing (lookup failure), the cloud-native router floods the packet out all the interfaces (except the incoming interface) in the bridge domain.

By default:

- MAC table entries time out after 60 seconds.
- The MAC table size is limited to 10,240 entries.

We recommend that you do not change the default values. Please contact Juniper Support if you need to change the default values.

You can see the MAC table entries by using:

- Introspect agent at **http://*host server IP*:8085/mac_learning.xml#Snh_FetchL2MacEntry**

l2_mac_entry_list

vrf_id	vlan_id	mac	index	packets	time_since_add	last_stats_change
0	1001	00:10:94:00:00:01	5644	615123154	12:55:14.248263	00:00:00.155450
0	1001	00:10:94:00:00:65	6480	615108294	12:55:14.247765	00:00:00.155461
0	1002	00:10:94:00:00:02	5628	615123173	12:55:14.248295	00:00:00.155470

- The command **show bridge mac-table** on the Cloud-Native Router controller CLI:

```
show bridge mac-table
Routing Instance : default-domain:default-project:ip-fabric:__default__
Bridging domain VLAN id : 3002
MAC                  MAC                  Logical
address              flags                interface

00:00:5E:00:53:01      D                   bond0
```

- The command **purel2cli --mac show** on the CLI of the vRouter pod:

```
purel2cli --mac show
==================================================
||  MAC            vlan      port      hit_count||
==================================================
00:01:01:01:01:03  1221      2          1101892
00:01:01:01:01:02  1221      2          1101819
00:01:01:01:01:04  1221      2          1101863
00:01:01:01:01:01  1221      2          1101879
5a:4c:4c:75:90:fe  1250      5          12
Total Mac entries 5
```

If you exceed the MAC address limit, the counter **pkt_drop_due_to_mactable_limit** increments. You can see this counter by using the introspect agent at **http://*host server IP*:8085/Snh_AgentStatsReq**.

If you delete or disable an interface, the cloud-native router deletes all the MAC entries associated with that interface from the MAC table.

MAC Entry Aging

The aging timeout for cached MAC entries is 60 seconds. You can configure the aging timeout at deployment time by editing the **values.yaml** file. The minimum timeout is 60 seconds and the maximum timeout is 10,240 seconds. You can see the time that is left for each MAC entry through introspect at

http:// *host server IP*:8085/mac_learning.xml#Snh_FetchL2MacEntry. We show an example of the output below:

```
l2_mac_entry_list
vrf_id          vlan_id         mac                  index          packets
time_since_add          last_stats_change
0               1001            00:10:94:00:00:01    5644           615123154
12:55:14.248785         00:00:00.155450
0               1001            00:10:94:00:00:65    6480           615108294
12:55:14.247765         00:00:00.155461
0               1002            01:10:94:00:00:02    5628           615123173
12:55:14.248295         00:00:00.155470
```

Storm Control

SUMMARY

Read this topic to understand how the broadcast rate limiting feature is implemented by the cloud-native router when deployed in L2 mode.

IN THIS SECTION

The storm control or rate limiting feature controls the rate of egress broadcast, unknown unicast, and multicast (BUM) traffic on fabric interfaces.

Configuration Example

You specify the rate limit in bytes per second by adjusting **stormControlProfiles** in the **values.yaml** file before deployment.

```
# rate limit profiles for bum traffic on fabric interfaces in bytes per second
 stormControlProfiles:
   rate_limit_pf1:
     bandwidth:
       level: 0
```

Once a profile is created, it can be assigned to the interface via the `storm-control-profile` interface attribute. For example:

```
- eth1:
      ddp: on
      interface_mode: trunk
      vlan-id-list: [100, 200, 300, 700-705]
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 100
      no-local-switching: true
```

The system applies the configured profiles to all specified fabric interfaces in the cloud-native router. The maximum per-interface rate limit value you can set is 1,000,000 bytes per second.

If the unknown unicast, broadcast, or multicast traffic rate exceeds the set limit on a specified fabric interface, the vRouter drops the traffic. You can see the drop counter values by running the `dropstats` command in the vRouter CLI. You can see the per-interface rate limit drop counters by running the vRouter CLI command `vif --get` *`fabric_vif_id`* `--get-drop-stats`. For example:

```
dropstats
L2 untag pkt drop             8832
L2 Src Mac lookup fail        880
Rate limit exceeded 29312474
```

When you configure a rate limit profile on a fabric interface, you can see the configured limit in bytes per second when you run either `vif --list` or `vif --get` *`fabric_vif_id`*.

```
vif0/2         PCI: 0000: af: 01.1 (Speed 10000, Duplex 1)
               Type: Physical HWaddr: 76:5d: f5: f5: c1:7a
               Vrf:0 Flags: L2Vof QOS:-1 Ref: 8 BUM Rate Limit: 1000000
               RX port    packets:1 errors:0
               RX queue packets:1 errors:0
               RX queue errors to lore 000000000000
               Driver: net_iavf
               Fabric Interface: 0000:af:01.1 Status: UP
               Vlan Mode: Trunk Vlan: 300 500 600
               RX packets:0  bytes:0
errors:1
               TX packets:0 bytes:0 errors:0
               Drops: 1
```

NOTE:

- The rate limit is only configurable on physical interfaces and only during deployment.
- The existing global rate limit configuration *fabricBMCastRateLimit* is deprecated from release 22.4.

APIs and CLI Commands for Bond Interfaces

SUMMARY

Read this topic to learn about the APIs and CLIs available in the L2 mode of the Juniper Cloud-Native Router. Cloud-Native Router supports an API that can be used to force traffic to switch from the active interface to the standby interface in a bonded pair. Another Cloud-Native Router API and a CLI can be used to view the active node details in a bond interface.

IN THIS SECTION

APIs for Bond Interfaces

When you run cloud-native router in L2 mode with cascaded nodes, you can configure those nodes to use bond interfaces. You can configure the bond mode in the `values.yaml` file before deployment. For example:

```
bondInterfaceConfigs:
    - name: "bond0"
      mode: 1             # ACTIVE_BACKUP MODE
      slaveInterfaces:
      - "enp59s0f0v0"
      - "enp59s0f0v1"
```

API to View the Active and Backup Interfaces in a Bond Interface Pair

Starting with Cloud-Native Router Release 23.3, use the REST API call: `curl -X GET http://127.0.0.1:9091/bond-get-active/bond0` on localhost port 9091 to fetch the active and backup interface details of a bond interface pair.

A sample output is shown below:

```
root@nodep23:~# curl -X GET http://127.0.0.1:9091/bond-get-active/bond0
{"active": "0000:af:01.0", "backup": "0000:af:01.1"}
```

API to Force Bond Link Switchover

Starting with Cloud-Native Router Release 22.4, you can force traffic switchover from an active to backup interface in a bond interface pair using a REST API. If you have configured the bond interface pair in the `ACTIVE_BACKUP` mode before deploying JCNR, then the vRouter-agent exposes the REST API call: `curl -X POST http://127.0.0.1:9091/bond-switch/bond0` on localhost port 9091. Use this REST API call to force traffic to switch from the active interface to the backup interface.

A sample output is shown below:

```
root@nodep23:~# curl -X GET http://127.0.0.1:9091/bond-get-active/bond0
{"active": "0000:af:01.0", "backup": "0000:af:01.1"}
root@nodep23:~# curl -X POST http://127.0.0.1:9091/bond-switch/bond0
{}
root@nodep23:~# curl -X GET http://127.0.0.1:9091/bond-get-active/bond0
{"active": "0000:af:01.1", "backup": "0000:af:01.0"}
```

CLI Commands for Bond Interfaces

The vRouter contains the following CLI commands which are related to bond interfaces:

- `dpdkinfo -b`—displays the active interface in a bonded pair.

```
[[root@jcnr-01 /]# dpdkinfo -b
No. of bond slaves: 2
Bonding Mode: Active Backup
Transmit Hash Policy: Layer 2 (Ethernet MAC)
```

```
MII status: UP
MII Link Speed: 10000 Mbps
Up Delay (ms): 0
Down Delay (ms): 0
Driver: net_bonding

Slave Interface(0): 0000:17:01.0
Slave Interface Driver: net_iavf
Slave Interface (0): Active
Slave Interface Mac : 6E: BD: 45:0F: 4A:02

MII status: UP
MII Link Speed: 10000 Mbps

Slave Interface (1): 0000:17:11.0
Slave Interface Driver: net_iavf
Slave Interface Mac      6E: BD: 45:0F: 4A: C2

MII status: UP
MII Link Speed: 25000 Mbps
```

- `dpdkinfo -n`—displays the traffic statistics associated with your bond interfaces.

```
[root@jcnr-01 /]# dpdkinfo -n2
Master Info (eth_bond_bond0):
RX Device Packets: 72019, Bytes: 96419113, Errors:0, Nombufs:0
Dropped RX Packets: 37475
TX Device Packets:0, Bytes:0, Errors:0
Queue Rx:
Tx:
Rx Bytes:
Tx Bytes:
Errors:

Slave Info (0000:17:01.0):
Rx Device Packets: 72019, Bytes:66073908, Errors:0, Nombufs:0
Dropped RX Packets: 588
TX Device Packets:0, Bytes:0, Errors:0
Queue Rx:
Tx:
Rx Bytes:
Tx Bytes:
```

```
Errors:

Slave Info (0000:17:11.0):
RX Device Packets:0, Bytes:30345205, Errors:0, Nombufs:0
Dropped R Packets:36887
TX Device Packets:0, Bytes:0, Errors:0
Queue Rx:
Tx:
Rx Bytes:
Tx Bytes:
Errors:
```

Native VLAN

IN THIS SECTION

Starting in Juniper Cloud-Native Router Release 23.1, Cloud-Native Router supports receiving and forwarding untagged packets on a trunk interface. Typically, trunk ports accept only tagged packets, and the untagged packets are dropped. You can enable a Cloud-Native Router fabric trunk port to accept untagged packets by configuring a native VLAN identifier (ID) on the interface on which you want the untagged packets to be received. When a Cloud-Native Router fabric trunk port is enabled to accept untagged packets, such packets are forwarded in the native VLAN domain.

Native VLAN

Enable the `native-vlan-id` key in the Helm chart, at the time of deployment, to configure the VLAN identifier and associate it with untagged data packets received on the fabric trunk interface. Edit the `values.yaml` file in **Juniper_Cloud_Native_Router_*<release-number>*/helmchart** directory and add the key `native-vlan-id` along with a value for it. For example:

```
fabricInterface:
  - eth1:
      ddp: on
      interface_mode: trunk
      vlan-id-list: [100, 200, 300, 700-705]
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 100
      no-local-switching: true
```

NOTE: After editing the **values.yaml** file, you have to install or upgrade Cloud-Native Router using the edited **values.yaml** to ensure that the `native-vlan-id` key is enabled.

To verify, if native VLAN is enabled for an interface, connect to the vRouter agent by executing the command `kubectl exec -it -n contrail contrail-vrouter-`*<agent container>* `-- bash` command, and then run the command `vif --get` *<interface index id>*. A sample output is shown below:

```
vif0/1      PCI: 0000:00:00.0 (Speed 10000, Duplex 1)
            Type:Physical HWaddr:6a:45:b2:a8:ce:5c
            Vrf:0 Flags:L2Vof QOS:-1 Ref:11
            RX port   packets:36550 errors:0
            RX queue  packets:36550 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond0  Status: UP  Driver: net_bonding
            Slave Interface(0): 0000:3b:02.0  Status: UP  Driver: net_iavf
            Vlan Mode: Trunk  Vlan: 100 200 300
            Native vlan id: 100
            RX packets:36550  bytes:5875795 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:613
```

Prevent Local Switching

IN THIS SECTION

Starting in Juniper Cloud-Native Router Release 23.1, Cloud-Native Router provides support to prevent interfaces in a bridge domain that are a part of the same VLAN group, from transmitting ethernet frame copies in between those interfaces. The **noLocalSwitching** key provides the option to enable the functionality on the selected VLAN IDs.

To prevent interfaces in a bridge domain from transmitting and receiving ethernet frame copies, enable the **noLocalSwitching** key and assign a VLAN ID to it to ensure that the interfaces belonging to the VLAN ID do not transmit frames to one another. Note that the **noLocalSwitching** functionality is enabled only on the access interfaces. To enable **noLocalSwitching** on a trunk interface that is a part of the same VLAN ID, you have to separately enable the trunk interface by setting the **no-local-switching** key in the trunk interface to **true**. Use the **noLocalSwitching** functionality when you want to block interfaces that are a part of a VLAN group to stop transmitting traffic directly to one another.

NOTE: For all the trunk interfaces and access interfaces, the cloud-native router isolates traffic for the bridge domains configured with **no-local-switching**.

Configuration Example

To prevent local switching, perform the steps below prior to the deploy time:

1. Edit the **values.yaml** file in **Juniper_Cloud_Native_Router_*<release-number>*/helmchart** directory.

2. Enable the **noLocalSwitching** key and provide the VLAN IDs.

```
noLocalSwitching: [700]
```

NOTE:

a. The value for the **noLocalSwitching** key can be an indivdual VLAN ID, or multipe comma-separated VLAN ID values, or a VLAN ID range, or a combination of comma-separated VLAN ID values and a VLAN ID range. For example, **noLocalSwitching: [700, 701, 705-710]**.

b. With this step the feature is enabled for all access interfaces having the specified VLAN ID. You can skip the next step if you do not want to enable the feature on the trunk interface.

3. To enable the feature on a trunk interface, add the key **no-local-switching** and set it to **true** under the trunk interface configuration.

   . For example:

```
fabricInterface:
  - bond0:
      ddp: on
      interface_mode: trunk
      vlan-id-list: [100, 200, 300, 700-705]
      storm-control-profile: rate_limit_pf1
      #native-vlan-id: 100
      no-local-switching: true
```

4. Install or upgrade Cloud-Native Router using the **values.yaml**.

Verify Configuration

To verify the configuration, you can use the `purel2cli` utility available on the vRouter. View the "Access vRouter CLI" on page 331 topic to access the vRouter shell. You can run the `purel2cli` commands from the vRouter CLI. For example:

1. Run the command `purel2cli --nolocal show` to know all the interfaces that are enabled for **noLocalSwitching** functionality on all the VLANs. A sample output is shown below:

```
[root@jcnr-01 /]# purel2cli --nolocal show
============================
vlan    no_local_switch_list
============================
100     1, 2, 4,
200
300
700
701
```

```
702
703
```

2. Run the command `purel2cli --nolocal get <VLAN ID>` to check if **noLocalSwitching** functionality is enabled on a specific VLAN ID. A sample output is shown below:

```
[root@jcnr-01 /]# purel2cli --nolocal get 100
============================
vlan    no_local_switch_list
============================
100     1, 2, 4,
```

Layer-2 VLAN Sub-Interfaces

IN THIS SECTION

VLAN sub-interfaces are like logical interfaces on a physical switch or router. They access only tagged packets that match the configured VLAN tag. A sub-interface has a parent interface. A parent interface can have multiple sub-interfaces, each with a VLAN ID. When you run the cloud-native router, you must associate each sub-interface with a specific VLAN. Starting in Juniper Cloud-Native Router Release 23.2, the cloud-native router supports the use of VLAN sub-interfaces in L3 mode along with the previously supported L2 mode.

Configuration Example

The VLAN sub-interfaces are configured using the Netowrk Attachment Definition (NAD) and pod YAML manifests. Please see the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 and relevant configuration examples for more information.

The Cloud-Native Router controller interface configuration viewed using the `show configuration` command is as shown below (the output is trimmed for brevity).

For L2 mode:

```
routing-instances {
    switch {
         instance-type virtual-switch;
         bridge-domains
{

             bd100 {
                 vlan-id 100;
                 interface vhostnet1-1e555ee1-7d93-40.100;
             }
         }
     }
}
```

On the vRouter, a VLAN sub-interface configuration is as shown below:

For L2 mode:

```
vif0/5      Virtual: vhostnet1-71cd7db1-1a5e-49.100 Vlan(o/i)(,S): 3003/3003 Parent:vif0/4
            Type:Virtual(Vlan) HWaddr:00:99:99:99:33:09
            Vrf:0 Flags:L2 QOS:-1 Ref:3
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

Enabling Dynamic Device Personalization (DDP) on Individual Interfaces

SUMMARY

Dynamic Device Personalization (DDP) is a technology that enables programmable packet processing pipeline provided by Intel as a profile to their NICs. Cloud-Native Router supports enabling Dynamic Device Personalization (DDP) on individual interfaces.

Starting with Juniper Cloud-Native Router (JCNR) Release 23.2, Cloud-Native Router supports enabling Dynamic Device Personalization (DDP) on individual interfaces. This feature is available on Cloud-Native Router in L2, L3, and L2-L3 modes.

Dynamic Device Personalization (DDP) is a technology that enables programmable packet processing pipeline provided by Intel as a profile to their NICs. Multiple Intel NICs support this technology. The support varies based on the Intel NIC type. DDP is used in packet classification where the profiles applied to the NIC can classify multiple packet formats on the NIC enabling speeds and feeds to the Data Plane Development Kit (DPDK).

Juniper cloud native router (JCNR) provides routing and switching functionality. Cloud-Native Router supports interfaces from different NIC cards. Some of the Intel NICs support DDP and some of them don't support DDP. Therefore, in a deployment scenario, Cloud-Native Router might have one interface from one NIC that supports DDP and another interface from a different NIC that does not support DDP. Cloud-Native Router supports enabling DDP per interface to overcome such issues.

NOTE: For E810 PF, Cloud-Native Router loads the DDP package which is bundled with JCNR. However, for other NICs, ensure you load the DDP package on the NICs before starting JCNR.

A DDP configuration is available per interface. This configuration option overrides global DDP (`ddp`) configuration for that interface. If you do not configure an interface DDP, then the global configuration value serves as the value for that interface. If you do not configure the global DDP configuration, then the default value for the global configuration which is `off` takes effect.

NOTE: DDP is supported on the following NICs:

- E810 VF
- E810 PF
- X710 PF
- XXV710 PF

DDP support is not available when interfaces are defined under subnets.

You should configure DDP in the helm chart before deployment. Configuring the DDP configurations in the helm charts for both global and at interface levels is optional. If you do not configure the DDP keys, then the default value for global DDP which is `off` takes effect.

The global DDP configuration is available in the `values.yaml` file as shown below:

```
# Set ddp to enable Dynamic Device Personalization (DDP)
# Provides datapath optimization at NIC for traffic like GTPU, SCTP etc.
# Options include auto or on or off; default: off
ddp: "auto"
```

You can configure one of the following options for `ddp` at the interface level:

1. Auto—when set to auto, Cloud-Native Router checks if the NIC supports DDP or not during deployment and configures DPDK accordingly. Detecting whether a NIC supports DDP at run time makes is easier to deploy Cloud-Native Router in volumes.
2. On—option enables DDP on the interface without validating the NIC. Use this option only if you are sure that the NIC supports DDP.
3. Off—is the default option at the interface level. This option disables DDP on the interface.

For example,

```
-    eth1:
        ddp: "off" ## auto or on or off
```

NOTE: Each interface can have a different configuration for `ddp`. DDP is enabled for a bond interface only if all the slave interface NICs support DDP.

3

CHAPTER

L3 Features

IN THIS CHAPTER

L3 Features Overview

SUMMARY

Read this topic to learn about the features available in the Juniper Cloud-Native Router when deployed in L3 (router) mode.

The Juniper Cloud-Native Router supports multiple "deployment modes" on page 13.

In L3 mode, the cloud-native router behaves like a router and so performs routing functions and runs routing protocols such as ISIS, BGP, OSPF, and segment routing-MPLS. In L3 mode, the pod network is divided into an IPv6 underlay network and an IPv4 or IPv6 overlay network. The IPv6 underlay network is used for control plane traffic. All L3 features are supported with DPDK datapath. Please review *Supported Cloud-Native Router Features for eBPF XDP* to verify feature support for eBPF XDP datapath.

This chapter provides information about the various L3 features supported by JCNR.

Link Layer Discovery Protocol (LLDP)

SUMMARY

Juniper Cloud-Native Router supports LLDP on Layer-3 interfaces to advertise capabilities, identity, and other information onto a LAN.

IN THIS SECTION

LLDP Overview

The Link Layer Discovery Protocol (LLDP) is an industry-standard method to enable networked devices to advertise capabilities, identity, and other information onto a LAN. The Juniper Cloud-Native Router supports LLDP on Layer 3 interfaces. LLDP information is sent at a fixed interval as an Ethernet frame.

Each frame contains one LLDP protocol data unit (PDU). Each LLDP PDU is a sequence of type-length-value (TLV) elements. Cloud-Native Router transmits the following mandatory and non-mandatory TLVs:

Table 2: Supported TLVs

Mandatory	Non-Mandatory
Chassis ID	Port Description
Port ID	System Name
Time to Live (TTL)	System Description
End TLV	System Capabilities (MAC/PHY configurations and status)
	Link Aggregation (Type 3 and Type 7)
	Max Frame Size

NOTE: By default Management-address TLV is not supported. The Management-address TLV is sent out only once the management address or management-interface is configured on Cloud-Native Router.

You can configure `tlv-filter` and `tlv-select` to filter or select non-mandatory TLVs. Cloud-Native Router will receive all TLVs support by Junos OS. You can view the received TLVs using the `show lldp neighbors` command output. Please review the LLDP Overview junos documentation for more details.

LLDP Configuration

You can configure the following options for the LLDP protocol in the Cloud-Native Router:

```
# set protocols lldp ?
Possible completions:
  advertisement-interval  Transmit interval for LLDP messages (5..32768 seconds)
+ apply-groups         Groups from which to inherit configuration data
+ apply-groups-except  Don't inherit configuration data from these groups
> chassis-id           Chassis-id to be used for Chassis ID TLV generation
```

```
  dest-mac-type        Destination address to be used
  disable              Disable LLDP
  fast-rx-processing   Start optimised processing of received pdu
  hold-multiplier      Hold timer interval for LLDP messages (2..10)
> interface            Interface configuration
  lldp-tx-fast-init    Transmission count in fast transmission mode (1..8)
  management-address   LLDP management address
  management-interface  Management interface to be used in LLDP PDUs
  mau-type             Populate mau-type in lldp PDU
  neighbour-port-info-display  Show lldp neighbors to display port-id or port-description
  port-description-type  The Interfaces Group MIB object to be used for Port Description TLV 
generation
  port-id-subtype      Sub-type to be used for Port ID TLV generation
  system-description   System description to be used in system-description TLV
  system-name          System name to be used in system-name TLV
+ tlv-filter           Filter TLVs to be sent
+ tlv-select           Select TLVs to be sent
> traceoptions         Trace options for LLDP
  transmit-delay       Transmit delay time interval for LLDP messages (1..8192 seconds)
```

Please review edit-protocols-lldp topic for more details about the configurable options.

You must perform LLDP configuration using a Configlet. Review *Customize JCNR Configuration* for more details. A sample configlet is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set protocols lldp enable
    set protocols lldp management-address 192.168.1.10
    set protocols lldp advertisement-interval 5
    set protocols lldp transmit-delay 1
    set protocols lldp hold-multiplier 2
    set protocols lldp chassis-id chassis-id-type mac-address
    set protocols lldp chassis-id chassis-id-value 2c:6b:f5:15:6b:c0
    set protocols lldp interface ens19
    set protocols lldp system-name jcnr1.ix.juniper.net
  crpdSelector:
```

```
    matchLabels:
      node: worker
```

LLDP Verification

You can verify the LLDP configuration and statistics using the cRPD `show` commands:

1. Verify the LLDP configuration:

```
user@host> show lldp

LLDP                      : Enabled
Advertisement interval    : 5 seconds
Transmit delay            : 1 seconds
Hold timer                : 10 seconds
Notification interval     : 5 Second(s)
Tx fast start count       : 1 Packets
Config Trap Interval      : 0 seconds
Connection Hold timer     : 300 seconds
Port ID TLV subtype       : locally-assigned
Port Description TLV type : interface-alias (ifAlias)

Interface      Parent Interface    LLDP        LLDP-MED       Power Negotiation
ens19          -                   Enabled
```

```
user@host> show lldp detail

LLDP                      : Enabled
Advertisement interval    : 5 seconds
Transmit delay            : 1 seconds
Hold timer                : 10 seconds
Notification interval     : 5 Second(s)
Tx fast start count      : 1 Packets
Config Trap Interval      : 0 seconds
Connection Hold timer     : 300 seconds
Port ID TLV subtype       : locally-assigned
Port Description TLV type : interface-alias (ifAlias)

Interface      Parent Interface    LLDP        LLDP-MED     Power Negotiation     Neighbor
```

```
count      Dest MAC
ens19          -                   Enabled
1                  01:80:C2:00:00:0E

Basic Management TLVs supported:
End Of LLDPDU, Chassis ID, Port ID, Time To Live, Port Description, System Name, System
Description, System Capabilities, Management Address

Organizationally Specific TLVs supported:
Port VLAN tag, Port VLAN name, MAC/PHY configuration/status, Link aggregation, Maximum Frame
Size
```

2. Verify LLDP local information of the local device:

```
user@host> show lldp local-information

LLDP Local Information details

Chassis ID   : 2c:6b:f5:15:6b:c0
Chassis type : Mac address
System name  : jcnr1.ix.juniper.net
System descr : Juniper Networks, Inc. crpd internet router, kernel JUNOS , Build date:
2024-11-19 12:16:34 UTC Copyright (c) 1996-2024
               Juniper Networks, Inc.

System Capabilities
    Supported       : Bridge Router
    Enabled         : Bridge Router

Management Information
    Interface Name  : ens19
    Address Subtype : IPv4(1)
    Address         : 192.168.1.10
    Interface Number               : 6
    Interface Numbering Subtype    : Unknown(1)

Interface name      Parent Interface   Interface ID       Interface description    Status
ens19               -                  6                  ens19                    Up
```

```
user@host> show lldp statistics
Interface    Parent Interface  Received  Unknown TLVs  With Errors  Discarded TLVs
```

```
Transmitted  Untransmitted
ens19        -                 458       0             0            0
486          0
```

3. Verify LLDP neighbors:

```
user@host> show lldp neighbors
Local Interface    Parent Interface    Chassis Id                               Port
info          System Name
ens19              -                   2c:6b:f5:16:6b:c0
ens20              jcnr2.ix.juniper.net
```

```
user@host> show lldp neighbors detail
LLDP Neighbor Information:
Local Information:
Index: 2 Time to live: 10 Time mark: Wed Nov 20 12:08:13 2024 Age: 4 secs
Local Interface    : ens19
Parent Interface   : -
Local Port ID      : 6
Ageout Count       : 1

Neighbor Information:
Chassis type       : Mac address
Chassis ID         : 2c:6b:f5:16:6b:c0
Port type          : Locally assigned
Port ID            : 13
Port description   : ens20
System name        : jcnr2.ix.juniper.net

System Description : Juniper Networks, Inc. crpd internet router, kernel JUNOS , Build date:
2024-11-19 12:16:34 UTC Copyright (c) 1996-2024 Juniper Networks, Inc.



System capabilities
        Supported: Bridge Router
        Enabled  : Bridge Router

Management address
        Address Type      : IPv4(1)
        Address           : 192.168.1.20
        Interface Number  : 0
```

```
        Interface Subtype : Unknown(1)
        OID               : 1.3.6.1.2.1.31.1.1.1.1.116.

Organization Info
       OUI      : IEEE 802.3 Private (0x00120f)
       Subtype  : MAC/PHY Configuration/Status (1)
       Info     : Autonegotiation [not supported, disabled (0x0)], PMD Autonegotiation
Capability (0x6c1d), MAU Type (0x0)
       Index    : 1

Organization Info
       OUI      : IEEE 802.3 Private (0x00120f)
       Subtype  : Link Aggregation (3)
       Info     : Aggregation Status [supported, disabled (0x1)], Aggregation Port ID (0)
       Index    : 2

Organization Info
       OUI      : IEEE 802.3 Private (0x00120f)
       Subtype  : Maximum Frame Size (4)
       Info     : MTU Size (1500)
       Index    : 3

Organization Info
       OUI      : Ethernet Bridged (0x0080c2)
       Subtype  : Link Aggregation 802.1  (7)
       Info     : Aggregation Status [supported, disabled, Aggregator Port %(0x1)],
Aggregation Port ID (0)
       Index    : 4
```

DHCP Relay

SUMMARY

Juniper Cloud-Native Router can relay DHCP messages between cascaded Next-Generation Distributed Units (NGDUs) and an external DHCP server.

Juniper Cloud-Native Router can be configured as a Stateless DHCP Relay agent for an L2-L3 deployment. It can relay DHCP messages between cascaded Next-Generation Distributed Units (NGDUs) and an external DHCP server. It supports simple packet forwarding non-snooping DHCPv4 and DHCPv6 relay feature between the DHCP client and DHCP server. It does not maintain leases or client state. When configured as a DHCPv4 relay agent, Cloud-Native Router is bypassed for subsequent lease renewals, once the client has obtained its address and configuration from the DHCP server. You can configure the same behavior for DHCPv6 implementation as well. In the forward-only implementation, the relay agent does not participate in the state exchange between the client and server. Hence, events such as reboot, Graceful Routing Engine switchover (GRES), or failover can quickly self-correct as the clients retry interrupted transactions.

Configuration

The following table lists the knobs and overrides that are supported for DHCPv4 and DHCPv6 relay options on Cloud-Native Router:

Table 3: DHCPv4 and DHCPv6 Support

Protocol	Supported Knobs	Supported Overrides

DHCPv4	`forward-only;`
	`relay-option-82`
		`always-write-option-82 (circuit-id | remote-id);`
		`relay-source;`
		`trust-option-82;`
		`user-defined-option-82` *`string`*

DHCPv6	`forward-only;`
	`relay-agent-interface-id`
	`relay-agent-remote-id`
		No DHPCv6 overrides supported

The configuration syntax for DHCPv4 relay agent is provided below. You can configure DHCPv4 relay agent under the `[edit]` and `[edit routing-instances]` hierarchy. Please review DHCP Relay CLI for command description and options.

```
[edit]
forwarding-options {
    dhcp-relay {
        active-server-group name;
        duplicate-clients-in-subnet (incoming-interface | option-82);
```

```
        forward-only;
        overrides {
            always-write-option-82  (circuit-id | remote-id);
            relay-source;
            trust-option-82;
            user-defined-option-82 string
        }
        relay-option-82 {
            circuit-id {
                prefix {
                    host-name;
                    logical-system-name;
                    routing-instance-name;
                }
                use-interface-description (device | logical);
                user-defined;
            }
            remote-id {
                prefix {
                    host-name;
                    logical-system-name;
                    routing-instance-name;
                }
                use-interface-description (device | logical);
            }
        }
        server-group name {
            ip-address;
        }
        group name {
            relay-option-82 {
                circuit-id {
                    prefix {
                        host-name;
                        logical-system-name;
                        routing-instance-name;
                    }
                    use-interface-description (device | logical);
                    user-defined;
                }
                remote-id {
                    prefix {
                        host-name;
```

```
                    logical-system-name;
                    routing-instance-name;
                }
                use-interface-description (device | logical);
            }
        }
        active-server-group name;
        interface interface_name;
    }
  }
}
```

NOTE: If a packet arrives with an `option-82` record and `trust-option-82` is not configured the packet will be dropped.

If a packet arrives with an `option-82` record while `relay-option-82` is configured, the original incoming `option-82` value is preserved with no changes.

The configuration syntax for DHCPv6 relay agent is provided below. You can configure DHCPv4 relay agent under the `[edit]` and `[edit routing-instances]` hierarchy. Please review DHCPv6 Relay CLI for command description and options.

```
[edit]
forwarding-options {
    dhcp-relay {
        dhcpv6 {
            active-server-group name;
            forward-only;
            relay-agent-interface-id {
                prefix {
                    host-name;
                    logical-system-name;
                    routing-instance-name;
                }
                use-interface-description (device | logical);
            }
            relay-agent-remote-id {
                prefix {
                    host-name;
                    logical-system-name;
                    routing-instance-name;
```

```
                }
                use-interface-description (device | logical);
            }
            server-group <name> {
                ip-address;
            }
            group name {
                relay-agent-interface-id {
                    prefix {
                        host-name;
                        logical-system-name;
                        routing-instance-name;
                    }
                    use-interface-description (device | logical);
                }
                relay-agent-remote-id {
                    prefix {
                        host-name;
                        logical-system-name;
                        routing-instance-name;
                    }
                    use-interface-description (device | logical);
                }
                active-server-group name;
                interface interface_name;
            }
        }
    }
}
```

You can configure DHCP tracing using the traceoptions configuration as shown in the snippet below:

```
[edit]
system {
    processes {
        dhcp-service {
            traceoptions {
                file jdhcpd size 20m;
                level all;
                flag all;
            }
        }
```

```
    }
}
```

Verification

You can verify the DHCP statistics via the "cRPD shell" on page 329.

Use `show dhcp statistics` to view DHCP service statistics.

```
root@controller-0> show dhcp statistics
Packets dropped:
    Total                      16
    No routing instance        16
```

Use `show dhcp relay statistics` to display DHCP relay statistics.

```
root@controller-0> show dhcp relay statistics
Packets dropped:
    Total                      16
    dhcp-service total         16

Messages received:
    BOOTREQUEST                0
    DHCPDECLINE                0
    DHCPDISCOVER               0
    DHCPINFORM                 0
    DHCPRELEASE                0
    DHCPREQUEST                0
    DHCPLEASEACTIVE            0
    DHCPLEASEUNASSIGNED        0
    DHCPLEASEUNKNOWN           0
    DHCPLEASEQUERYDONE         0
    DHCPACTIVELEASEQUERY       0

Messages sent:
    BOOTREPLY                  0
    DHCPOFFER                  0
    DHCPACK                    10
    DHCPNAK                    0
    DHCPFORCERENEW             2
    DHCPLEASEQUERY             2
```

```
    DHCPBULKLEASEQUERY        0
    DHCPLEASEACTIVE           0
    DHCPLEASEUNASSIGNED       0
    DHCPLEASEUNKNOWN          0
    DHCPLEASEQUERYDONE        0
    DHCPACTIVELEASEQUERY      7
```

Use `show dhcpv6 relay statistics` to view DHCPv6 relay statistics.

```
root@controller-0> show dhcpv6 relay statistics
Dhcpv6 Packets dropped:
    Total              0

Messages received:
    DHCPV6_DECLINE            0
    DHCPV6_SOLICIT            2
    DHCPV6_INFORMATION_REQUEST 0
    DHCPV6_RELEASE            0
    DHCPV6_REQUEST            2
    DHCPV6_CONFIRM            0
    DHCPV6_RENEW              0
    DHCPV6_REBIND             0
    DHCPV6_RELAY_FORW         0
    DHCPV6_LEASEQUERY_REPLY   0
    DHCPV6_LEASEQUERY_DATA    0
    DHCPV6_LEASEQUERY_DONE    0
    DHCPV6_ACTIVELEASEQUERY   0

Messages sent:
    DHCPV6_ADVERTISE          0
    DHCPV6_REPLY              0
    DHCPV6_RECONFIGURE        0
    DHCPV6_RELAY_REPLY        0
    DHCPV6_LEASEQUERY         0
    DHCPV6_LEASEQUERY_REPLY   2
    DHCPV6_LEASEQUERY_DATA    0
    DHCPV6_LEASEQUERY_DONE    0
    DHCPV6_ACTIVELEASEQUERY   0
```

You can clear the DHCP statistics using the commands provided below:

```
clear dhcp statistics
clear dhcp relay statistics
clear dhcpv6 relay statistics
```

Layer-3 Class of Service (CoS)

SUMMARY

Juniper Cloud-Native Router supports Layer 3 Class of Service (CoS), also known as L3 Quality of Service (QoS). This topic provides an overview of the supports CoS mechanisms followed by configuration and verification examples.

IN THIS SECTION

L3 CoS Overview

IN THIS SECTION

When a network experiences congestion and delay, some packets must be prioritized to avoid random loss of data. Class of service (CoS), also known as Quality of Service (QoS) accomplishes this prioritization by dividing similar types of traffic, such as e-mail, streaming video, voice, large document file transfer, into classes. You then apply different levels of priority, such as those for throughput and packet loss, to each group, and thereby control traffic behavior. Juniper Cloud-Native Router supports CoS, enabling it to differentiate or classify traffic. It can either drop traffic or lower the priority of the traffic as per the rules configured. You can read more about Class of Service in the Junos documentation.

The Cloud-Native Router CoS application supports DiffServ, which uses a 6-bit differentiated services code point (DSCP) in the differentiated services field of the IPv4 and IPv6 packet header. For IPv6, DSCP is referred to as traffic class. The configuration uses DSCP values to decide the CoS treatment for the incoming packet. The DSCP field is also used to notify the modified priority of a packet to the next hop.

Cloud-Native Router Supported CoS Mechanisms

Cloud-Native Router supports Classifier, Policer and Rewrite/Marker CoS mechanisms. Let us look at the Cloud-Native Router CoS implementations in detail in the sections below.

Forwarding Classes and Loss Priority

The forwarding classes affect the forwarding and marking policies applied to packets as they transit JCNR. By default Cloud-Native Router has no forwarding classes defined. You can define up to 16 custom forwarding classes mapped to 8 queues.

In Cloud-Native Router CoS implementation forwarding class along with the loss priority is used for rewrite rules and policer. Loss priority is set by the classifier as low, medium low, medium high and high. Here are some of the ways loss priority is used in the Cloud-Native Router CoS implementation:

Table 4: Using Forwarding Class and Loss Priority

QoS Block	How Loss priority is used
Classifier	Forwarding class and loss priority are set by the classifier.
Rewrite	Loss priority is used as an index along with the Traffic Class (Forwarding Class) to obtain a new DSCP value.
Policer	Only color aware policer uses loss priority. Loss priority maps to traffic colors as follows: • Loss priority Low is mapped to Green • Loss priority Medium High and Medium Low are mapped to Yellow • Loss priority High is mapped to Red
Scheduler	Forwarding class and loss priority are inputs to the scheduler for queuing the traffic based on strict priority.

Classifiers

Packet classification refers to the examination of an incoming packet. This function associates the packet with a particular CoS servicing level. Classifiers associate incoming packets with a forwarding class and loss priority and, based on the associated forwarding class, assign packets to output queues. Two general types of classifiers are supported:

- Behavior Aggregate Classifier—Behavior aggregate (BA) is a method of classification that operates on a packet as it enters JCNR. The CoS value in the packet header is examined, and this single field determines the CoS settings applied to the packet. BA classifiers allow you to set the forwarding class and loss priority of a packet based on the Differentiated Services code point (DSCP) value and DSCP IPv6 value. BA classifier is configured at the interface level. You can read more about Behavior Aggregate Classifer in the Junos documentation.

- Multifield Traffic Classifier—A multifield (MF) classifier can examine multiple fields in the packet, such as the source and destination address of the packet as well as the source and destination port numbers. With multifield classifiers, you set the forwarding class and loss priority of a packet based on firewall filter (ACL) rules. If a packet matches both BA and MF classifiers, MF classifier takes precedence. You can read more about the Multified Traffic Classifier in the Junos documentation.

Policers

Policers allow you to limit traffic of a certain class to a specified bandwidth and burst size. Policer meters (measures) each packet against traffic rates and burst sizes configured. It then passes the packet and the metering result to the marker (rewrite rules), which assigns a packet loss priority that corresponds to the metering result. Based on the particular set of traffic limits configured, a policer identifies a traffic flow as belonging to one of either two or three categories that are similar to the colors of a traffic light used to control automobile traffic. Policers can be applied per packet or per traffic class. Cloud-Native Router CoS implementation supports 16 policer profiles. You can read more about Policer Implementation in Junos documentation. In Cloud-Native Router CoS implementation policers work in Single-rate three color marker (srTCM) or Two-rate three-color marker (trTCM) mode. Policers can perform traffic color marking in color aware or color blind modes. In color aware mode, the policer considers the packet's color as derived by classifier as additional input. In color blind mode the policer does not take into consideration packet's color while determining the new color. We will look at the different considerations for each of the modes in the tables provided hereafter.

- Single-rate three-color—A Single Rate Three Color Marker (srTCM) meters traffic based on the configured committed information rate (CIR), committed burst size (CBS), and the peak burst size (PBS). A single-rate three-color policer is most useful when a service is structured according to packet length and not peak arrival rate. Traffic is marked as belonging to one of three categories—green, yellow, or red based on the following considerations:

Table 5: Color Aware srTCM

Incoming Color	Packet Metered Against	Possible Cases	New Color	Action on the packet
Green	CIR, CBS, PBS	Below CBS	Green	Not dropped
		Above CBS but below PBS	Yellow	Change the traffic class using the rewrite rules.
		Above PBS	Red	Discard
Yellow	PBS	Below PBS	Yellow	Change the traffic class using the rewrite rules.
		Above PBS	Red	Discard
Red	Not metered	NA	Red	Discard

Table 6: Color Blind srTCM

Packet Metered Against	Possible Cases	New Color	Action on the packet
CIR, CBS, PBS	Below CBS	Green	Not dropped
	Above CBS but below PBS	Yellow	Change the traffic class using the rewrite rules.
	Above PBS	Red	Discard

- Two-rate three-color—A Two Rate Three Color Marker (trTCM) meters traffic based on the configured CIR and peak information rate (PIR). A two-rate three-color policer is most useful when a service is structured according to arrival rates and not necessarily packet length. Traffic is marked as belonging to one of three categories based on the following considerations:

Table 7: Color Aware trTCM

Incoming Color	Packet Metered Against	Possible Cases	New Color	Action on the packet
Green	CIR, PIR	Below CIR	Green	Not dropped
		Above CIR but below PIR	Yellow	Change the traffic class using the rewrite rules.
		Above PIR	Red	Discard
Yellow	PIR	Below PIR	Yellow	Change the traffic class using the rewrite rules.
		Above PIR	Red	Discard
Red	Not metered	NA	Red	Discard

Table 8: Color Blind trTCM

Packet Metered Against	Possible Cases	New Color	Action on the packet
CIR, PIR	Below CIR	Green	Not dropped
	Above CIR but below PIR	Yellow	Change the traffic class using the rewrite rules.
	Above PIR	Red	Discard

The colors marked by Policer are mapped to loss priority as follows:

Table 9: Mapping Colors to Loss Priority

Color	Loss Priority
Green	Low
Yellow	Medium-Low (Medium-High is not used while mapping color to loss priority)
Red	High

Rewrite/Marker

A rewrite rule or marker sets the appropriate CoS bits in the outgoing packet. This allows the next downstream routing device to classify the packet into the appropriate service group. Rewriting, or marking, outbound packets is useful when the routing device is at the border of a network and must alter the CoS values to meet the policies of the targeted peer. A rewrite profile is applied to the interface and is based on the forwarding class and loss priority derived by the classifier and/or the metering results of the policer. Cloud-Native Router rewrite rules supports copying outer IPv4/IPv6 DSCP marking to inner IP header DSCP field as well as MPLS EXP bit marking. Cloud-Native Router supports 16 rewrite profiles. You can read more about Rewrite rules in Junos documentation.

NOTE: Cloud-Native Router CoS implementation does not support implicit best effort treatment for traffic that does not match any CoS block. An explicit catch-all rule must be configured for best effort treatment.

NOTE: Cloud-Native Router CoS implementation modifies the DSCP value of the outer IP header only for tunneled packets (MPLSoUDP and VXLAN).

Scheduler

Scheduler is the last block in Cloud-Native Router's CoS implementation and computes the priority of the packets. Cloud-Native Router implements a strict priority 8-queue scheduler, with priority order high to low. The forwarding class is directly mapped to scheduler priority. The Cloud-Native Router supports a maximum of 4 scheduler profiles in the deployment helm chart and a maximum of 16 scheduler maps and forwarding classes. You can read more about Scheduler in Junos documentation. You can configure a scheduler with one of 8 priorities as provided in the below table. Note that the scheduler priorities are already mapped to interface queues (8 queues).

Table 10: Scheduler Priorities

Priority	Scheduling Priority (Queue)
high	Scheduling priority 1
low	Scheduling priority 7 (Least)
low-high	Scheduling priority 5
low-latency	Scheduling priority 4
low-medium	Scheduling priority 6
medium-high	Scheduling priority 2
medium-low	Scheduling priority 3
strict-high	Scheduling priority 0 (Highest)

Dropper and Shaper

The scheduler blocks also includes dropper and shaper modules.

- Dropper—The DPDK dropper module drops packets arriving at the scheduler block to avoid congestion. The drop is performed based on the weighted random early detection (WRED) drop profile maps configured. In the event of severe congestion, the dropper module may perform tail drop, resulting in dropping all arriving packets. The drop profile is applied per scheduler queue and may be based on packet loss priority. A maximum of 32 drop profiles and 3 drop profile maps per scheduler are supported. You can read more about dropper in Junos documentation.
- Shaper—The shaper or transmit rate is used to shape traffic per egress queue. By limiting the queue transmit rate, shaper can prevent high priority queues from starving low priority queues. Shaping rate is applied on the scheduler queues.

Supported Interfaces

Cloud-Native Router supports the following interfaces for CoS implementation:

Table 11: Support Interfaces for CoS Implementation

CoS Component/ Interface Type	Pod Interface	Fabric Interface	IRB Interface
Classifier	Supported	Supported	Supported
Policer	Supported	Supported	Supported
Marker	Supported	Supported	Supported
Scheduler	Not Supported	Supported	Not Supported

L3 CoS Configuration

You must configure a CoS scheduler profile per interface in the helm chart that defines the number of scheduler lcores and bandwidth. Review *Helm Chart customization* for more details.

You must perform CoS configuration on the Cloud-Native Router control plane using a Configlet. Review *Customize Cloud-Native Router Configuration* for more details. Sample configlets are provided below:

Configure Forwarding Classes

Cloud-Native Router does not have any forwarding classes configured by default. You can configure upto 16 forwarding classes, mapped to 8 queues. The user-defined queue mapping is ignored since queue numbers are derived from the priority defined in the scheduler configuration, mapped one-on-one with the queue number:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service forwarding-classes class af11 queue-num 3
    set class-of-service forwarding-classes class af12 queue-num 3
    set class-of-service forwarding-classes class af13 queue-num 3
    set class-of-service forwarding-classes class af21 queue-num 4
```

```
    set class-of-service forwarding-classes class af22 queue-num 4
    set class-of-service forwarding-classes class af23 queue-num 4
    set class-of-service forwarding-classes class af31 queue-num 5
    set class-of-service forwarding-classes class af32 queue-num 5
    set class-of-service forwarding-classes class af33 queue-num 5
    set class-of-service forwarding-classes class cs1 queue-num 6
    set class-of-service forwarding-classes class cs2 queue-num 6
    set class-of-service forwarding-classes class cs3 queue-num 7
    set class-of-service forwarding-classes class cs4 queue-num 7
    set class-of-service forwarding-classes class ef queue-num 1
    set class-of-service forwarding-classes class nc1 queue-num 0
    set class-of-service forwarding-classes class nc2 queue-num 0
  crpdSelector:
    matchLabels:
      node: worker
```

Configure Classifiers

You can configure a BA classifier and apply it on an interface:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service classifiers dscp BA_1 forwarding-class af11 loss-priority high code-
points 000001
    set class-of-service classifiers dscp BA_1 forwarding-class af11 loss-priority low code-
points 001010
    set class-of-service classifiers dscp BA_1 forwarding-class af11 loss-priority medium-high
code-points 001110
    set class-of-service classifiers dscp BA_1 forwarding-class af11 loss-priority medium-low
code-points 001100
    set class-of-service classifiers dscp BA_1 forwarding-class ef loss-priority low code-points
101111
    set class-of-service classifiers dscp BA_1 forwarding-class nc1 loss-priority low code-
points 110000
    set class-of-service classifiers dscp BA_1 forwarding-class nc1 loss-priority low code-
points 111000
```

```
    set class-of-service classifiers dscp-ipv6 BA6_1 forwarding-class ef loss-priority low code-
points 101111
    set class-of-service classifiers dscp-ipv6 BA6_1 forwarding-class nc2 loss-priority high
code-points 111000
    set class-of-service interfaces eno2 unit 0 classifiers dscp BA_1
    set class-of-service interfaces eno2 unit 0 classifiers dscp-ipv6 BA6_1
  crpdSelector:
    matchLabels:
      node: worker
```

You can configure MF classifier as a firewall filter:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set firewall family inet filter MCFF_1 term 1 from source-port 1001
    set firewall family inet filter MCFF_1 term 1 then forwarding-class af11
    set firewall family inet filter MCFF_1 term 1 then loss-priority medium-low
    set firewall family inet filter MCFF_1 term 2 from icmp-type echo-request
    set firewall family inet filter MCFF_1 term 2 then forwarding-class af31
    set firewall family inet filter MCFF_1 term 2 then loss-priority medium-low
    set firewall family inet filter MCFF_1 term 3 then accept
    set firewall family inet6 filter MCFF6_1 term 1 from source-port 1001
    set firewall family inet6 filter MCFF6_1 term 1 then forwarding-class af11
    set firewall family inet6 filter MCFF6_1 term 1 then loss-priority low
    set firewall family inet6 filter MCFF6_1 term 2 then accept
    set interfaces eno2 unit 0 family inet filter input MCFF_1
    set interfaces eno2 unit 0 family inet6 filter input MCFF6_1
  crpdSelector:
    matchLabels:
      node: worker
```

Configure Policers

You can configure the policer with the action type `three-color-policer` as follows:

Single Rate Three Color Meter (srTCM) Policer:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set firewall three-color-policer srTCM_1 action loss-priority high then discard
    set firewall three-color-policer srTCM_1 single-rate color-aware
    set firewall three-color-policer srTCM_1 single-rate committed-information-rate 500m
    set firewall three-color-policer srTCM_1 single-rate committed-burst-size 20k
    set firewall three-color-policer srTCM_1 single-rate excess-burst-size 20k
    set firewall family inet filter FF_1 term 1 then three-color-policer single-rate srTCM_1
    set firewall family inet6 filter FF6_1 term 1 then three-color-policer single-rate srTCM_1
  crpdSelector:
    matchLabels:
      node: worker
```

Two Rate Three Color Meter (trTCM) Policer:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set firewall three-color-policer trTCM_1 action loss-priority high then discard
    set firewall three-color-policer trTCM_1 two-rate color-aware
    set firewall three-color-policer trTCM_1 two-rate committed-information-rate 100m
    set firewall three-color-policer trTCM_1 two-rate committed-burst-size 2048
    set firewall three-color-policer trTCM_1 two-rate peak-information-rate 200m
    set firewall three-color-policer trTCM_1 two-rate peak-burst-size 2048
    set firewall family inet filter FF_1 term 1 from forwarding-class af11
    set firewall family inet filter FF_1 term 1 then three-color-policer two-rate trTCM_1
    set firewall family inet6 filter FF6_1 term 1 from forwarding-class af11
    set firewall family inet6 filter FF6_1 term 1 then three-color-policer two-rate trTCM_1
  crpdSelector:
```

```
    matchLabels:
      node: worker
```

You can configure classifiers with policers.

BA classifier with policer:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set firewall family inet filter FF_1 term 1 from forwarding-class af11
    set firewall family inet filter FF_1 term 1 then three-color-policer single-rate srTCM_1
    set firewall family inet6 filter FF6_1 term 1 from forwarding-class af11
    set firewall family inet6 filter FF6_1 term 1 then three-color-policer single-rate srTCM_1
    set class-of-service interfaces eno2 unit 0 classifiers dscp BA_1
    set class-of-service interfaces eno2 unit 0 classifiers dscp-ipv6 BA6_1
    set interfaces eno2 unit 0 family inet filter input FF_1
    set interfaces eno2 unit 0 family inet6 filter input FF6_1
  crpdSelector:
    matchLabels:
      node: worker
```

MF classifier as a firewall filter with policer:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set firewall family inet filter MCFF_1 term 1 from source-port 1001
    set firewall family inet filter MCFF_1 term 1 then forwarding-class af11
    set firewall family inet filter MCFF_1 term 1 then three-color-policer single-rate srTCM_1
    set firewall family inet filter MCFF_1 term 2 from icmp-type echo-request
    set firewall family inet filter MCFF_1 term 2 then forwarding-class af31
    set firewall family inet filter MCFF_1 term 2 then three-color-policer single-rate srTCM_1
    set firewall family inet filter MCFF_1 term 3 then accept
```

```
    set firewall family inet6 filter MCFF6_1 term 1 from source-port 1001
    set firewall family inet6 filter MCFF6_1 term 1 then forwarding-class af11
    set firewall family inet6 filter MCFF6_1 term 1 then three-color-policer single-rate srTCM_1
    set firewall family inet6 filter MCFF6_1 term 2 then accept
    set interfaces eno2 unit 0 family inet filter input MCFF_1
    set interfaces eno2 unit 0 family inet6 filter input MCFF6_1
  crpdSelector:
    matchLabels:
      node: worker
```

NOTE: When the same policer is configured for multiple firewall filter terms, multiple policer instances are created, one per term.

Configure Rewrite/Marker

You can configure the rewrite/marker as follows:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service rewrite-rules dscp RE_1 forwarding-class af11 loss-priority high code-
point 010010
    set class-of-service rewrite-rules dscp RE_1 forwarding-class af11 loss-priority low code-
point 001100
    set class-of-service rewrite-rules dscp RE_1 forwarding-class af11 loss-priority medium-high
code-point 000001
    set class-of-service rewrite-rules dscp RE_1 forwarding-class af11 loss-priority medium-low
code-point 001110
    set class-of-service rewrite-rules dscp RE_1 forwarding-class ef loss-priority low code-
point 001010
    set class-of-service rewrite-rules dscp RE_1 forwarding-class ef loss-priority medium-high
code-point 001001
    set class-of-service rewrite-rules dscp-ipv6 RE6_1 forwarding-class af11 loss-priority high
code-point 010010
    set class-of-service rewrite-rules dscp-ipv6 RE6_1 forwarding-class af11 loss-priority low
code-point 001100
```

```
    set class-of-service rewrite-rules dscp-ipv6 RE6_1 forwarding-class af11 loss-priority
medium-high code-point 000001
    set class-of-service rewrite-rules dscp-ipv6 RE6_1 forwarding-class af11 loss-priority
medium-low code-point 001110
    set class-of-service rewrite-rules dscp-ipv6 RE6_1 forwarding-class ef loss-priority low
code-point 001001
    set class-of-service interfaces eno3 unit 0 rewrite-rules dscp RE_1
    set class-of-service interfaces eno3 unit 0 rewrite-rules dscp-ipv6 RE6_1
  crpdSelector:
    matchLabels:
      node: worker
```

MPLS Exp Rewrite for tunnel packets:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service rewrite-rules exp RE_3 forwarding-class af11 loss-priority low code-
point 111
    set class-of-service rewrite-rules exp RE_3 forwarding-class af11 loss-priority medium-high
code-point 101
    set class-of-service rewrite-rules exp RE_3 forwarding-class af11 loss-priority medium-low
code-point 110
    set class-of-service interfaces eno3 unit 0 rewrite-rules exp RE_3
  crpdSelector:
    matchLabels:
      node: worker
```

Configure Schedulers

You can configure the schedulers to one of 8 priorities. The scheduler map configures the forwarding class to scheduler mapping and is applied at the interface level.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
```

```
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service schedulers S1 priority medium-high
    set class-of-service schedulers S2 priority strict-high
    set class-of-service schedulers S3 priority high
    set class-of-service schedulers S4 priority low
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class af11 scheduler S1
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class af12 scheduler S2
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class ef scheduler S3
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class nc1 scheduler S4
    set class-of-service interfaces eno2 scheduler-map VOICE-SCHED-MAP
  crpdSelector:
    matchLabels:
      node: worker
```

NOTE: The CoS configuration is valid only if the fabric interface is mapped to a `qos-scheduler-profile` in the helm chart. The scheduler profile must be configured with CPU and bandwidth allocation. Please see *Helm Chart customization* for more details.

Configuring Dropper and Shaper

You can configure dropper and shaper modules within the scheduler block. Define a drop profile map and assign it to the scheduler along with the transmit rate.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set class-of-service drop-profiles dp1 min-threshold 25
    set class-of-service drop-profiles dp1 max-threshold 75
    set class-of-service drop-profiles dp1 mark-probablility-denominator 90
    set class-of-service drop-profiles dp2 min-threshold 35
    set class-of-service drop-profiles dp2 max-threshold 60
    set class-of-service drop-profiles dp2 mark-probablility-denominator 80
    set class-of-service schedulers S1 drop-profile-map loss-priority low protocol any drop-
```

```
profile dp2
    set class-of-service schedulers S1 drop-profile-map loss-priority high protocol any drop-
profile dp2
    set class-of-service schedulers S1 drop-profile-map loss-priority medium-high protocol any
drop-profile dp1
    set class-of-service schedulers S1 priority low
    set class-of-service schedulers S1 transmit-rate 50M
    set class-of-service schedulers S2 drop-profile-map loss-priority any protocol any drop-
profile dp1
    set class-of-service schedulers S2 priority high
    set class-of-service schedulers S2 transmit-rate 50M
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class af11 scheduler S1
    set class-of-service scheduler-maps VOICE-SCHED-MAP forwarding-class af12 scheduler S2
    set class-of-service interfaces eno2 scheduler-map VOICE-SCHED-MAP
  crpdSelector:
    matchLabels:
      node: worker
```

L3 CoS Verification

IN THIS SECTION

CLI commands to verify L3 CoS configuration

Verify Classifier Configuration on cRPD

You can verify the L3 CoS configuration using the following commands on the cRPD shell:

- Verify the classifier configuration using the `show class-of-service classifier` command. An example output is provided below:

```
user@host> show class-of-service classifier

Classifier: BA_1, Code point type: dscp
```

```
Code point              Forwarding class              Loss priority
000001                  af11                          high
001000                  cs1                           medium-high
001010                  af11                          low
001100                  af11                          medium-low
001110                  af11                          medium-high
010000                  cs1                           medium-high
010010                  af21                          medium-low
010100                  af22                          medium-high
010110                  af23                          high
011100                  af32                          high
101111                  ef                            low
110000                  nc1                           low
111000                  nc1                           low
111010                  af31                          low

Classifier: BA6_1, Code point type: dscp-ipv6
Code point              Forwarding class              Loss priority
001010                  af33                          low
010000                  cs2                           medium-high
011000                  cs3                           medium-low
101111                  ef                            low
110000                  cs4                           medium-low
111000                  nc2                           high
```

- Verify the code-point-alisases configuration using the `show class-of-service code-point-aliases` command. An example output is provided below:

```
user@host> show class-of-service code-point-aliases

Code point type: dscp
Alias                       Bit pattern
af11                        001010
af12                        001100
af13                        001110
af21                        010010
af22                        010100
af23                        010110
af31                        011010
af32                        011100
af33                        011110
af41                        100010
```

```
af42                            100100
af43                            100110
be                              000000
cs1                             001000
cs2                             010000
cs3                             011000
cs4                             100000
cs5                             101000
cs6                             110000
cs7                             111000
ef                              101110
nc1                             110000
nc2                             111000

Code point type: dscp-ipv6
  Alias                 Bit pattern
af11                            001010
af12                            001100
af13                            001110
af21                            010010
af22                            010100
af23                            010110
af31                            011010
af32                            011100
af33                            011110
af41                            100010
af42                            100100
af43                            100110
be                              000000
cs1                             001000
cs2                             010000
cs3                             011000
cs4                             100000
cs5                             101000
cs6                             110000
cs7                             111000
ef                              101110
nc1                             110000
nc2                             111000
```

- Verify the CoS configuration on an interface using the `show class-of-service interface` command. An example output is provided below:

```
user@host> show class-of-service interface eno2
Physical interface: eno2
Maximum usable queues: 8, Queues in use: 8

  Logical interface: eno2.0
Object                 Name                   Type
Classifier             BA_1                   dscp
Classifier             BA6_1                  dscp-ipv6
```

- Verify the rewrite rule configuration using the `show class-of-service rewrite-rule` command. Example outputs are provided below:

```
user@host> show class-of-service rewrite-rule

Rewrite rule: RE_1, Code point type: dscp
Forwarding class            Loss priority        Code point
ef                          low                  100001
af11                        low                  000010
af11                        high                 000101
af11                        medium-low           000011
af11                        medium-high          000100
af21                        medium-low           000110
af22                        medium-high          000111
af23                        high                 001000
af31                        low                  001111
af32                        high                 001010
cs1                         medium-high          001011
nc1                         low                  001101

Rewrite rule: RE6_1, Code point type: dscp-ipv6
Forwarding class            Loss priority        Code point
ef                          low                  000001
af33                        low                  001110
cs2                         medium-high          010011
cs3                         medium-low           101010
cs4                         medium-low           111111
nc2                         high                 111100
```

```
Rewrite rule: RE_3, Code point type: exp
Forwarding class            Loss priority        Code point
af11                        low                  111
af11                        medium-low           110
af11                        medium-high          101
```

- Verify the scheduler, dropper and shaper configuration using the `show class-of-service schedulers` command. Example output is provided below:

```
user@host> show class-of-service schedulers
S1 {
    transmit-rate 50m;
    drop-profile-map loss-priority low protocol any drop-profile dp2;
    drop-profile-map loss-priority high protocol any drop-profile dp2;
    drop-profile-map loss-priority medium-high protocol any drop-profile dp1;
    priority low;
}
S2 {
    transmit-rate 50m;
    drop-profile-map loss-priority any protocol any drop-profile dp1;
    priority high;
}
```

Verify CoS Classifiers on vRouter

```
bash-5.1# qosdpdk --list cla
Classifer name: BA_1 Traffic-class: dscp
================================================================
code-points          loss priority         forwarding-class-id
================================================================
   000000(0)            n/a                           n/a
   000001(1)            high                          65
   000010(2)            n/a                           n/a
   000011(3)            n/a                           n/a
   000100(4)            n/a                           n/a
   000101(5)            n/a                           n/a
   000110(6)            n/a                           n/a
   000111(7)            n/a                           n/a
   001000(8)            medium-high                   74
   001001(9)            n/a                           n/a
```

```
   001010(10)          low                         65
   001011(11)          n/a                         n/a
   001100(12)          medium-low                  65
   001101(13)          n/a                         n/a
   001110(14)          medium-high                 65
   001111(15)          n/a                         n/a
   010000(16)          medium-high                 74
   010001(17)          n/a                         n/a
   010010(18)          medium-low                  68
   010011(19)          n/a                         n/a
   010100(20)          medium-high                 69
   010101(21)          n/a                         n/a
   010110(22)          high                        70
   010111(23)          n/a                         n/a
   011000(24)          n/a                         n/a
   011001(25)          n/a                         n/a
   011010(26)          n/a                         n/a
   011011(27)          n/a                         n/a
   011100(28)          high                        72
   011101(29)          n/a                         n/a
   011110(30)          n/a                         n/a
   011111(31)          n/a                         n/a

======================================================================



Classifer name: BA6_1 Traffic-class: dscpv6
===============================================================
code-points          loss priority         forwarding-class-id
===============================================================
   000000(0)           n/a                         n/a
   000001(1)           n/a                         n/a
   000010(2)           n/a                         n/a
   000011(3)           n/a                         n/a
   000100(4)           n/a                         n/a
   000101(5)           n/a                         n/a
   000110(6)           n/a                         n/a
   000111(7)           n/a                         n/a
   001000(8)           n/a                         n/a
   001001(9)           n/a                         n/a
   001010(10)          low                         73
   001011(11)          n/a                         n/a
   001100(12)          n/a                         n/a
```

```
   001101(13)          n/a                            n/a
   001110(14)          n/a                            n/a
   001111(15)          n/a                            n/a
   010000(16)          medium-high                    75
   010001(17)          n/a                            n/a
   010010(18)          n/a                            n/a
   010011(19)          n/a                            n/a
   010100(20)          n/a                            n/a
   010101(21)          n/a                            n/a
   010110(22)          n/a                            n/a
   010111(23)          n/a                            n/a
   011000(24)          medium-low                     76
   011001(25)          n/a                            n/a
   011010(26)          n/a                            n/a
   011011(27)          n/a                            n/a
   011100(28)          n/a                            n/a
   011101(29)          n/a                            n/a
   011110(30)          n/a                            n/a
   011111(31)          n/a                            n/a

======================================================================
```

Verify Rewrite Rules

```
bash-5.1# qosdpdk --list re-write
Re-Write name: RE_1 Traffic-class: dscp
======================================================================
loss priority            Forwarding-class           re-write prio
======================================================================
   low                         0                          001100(12)
   low                         1                          n/a
   low                         2                          n/a
   low                         3                          101111(47)
   low                         4                          001010(10)
   low                         5                          010100(20)
   low                         6                          111010(58)
   low                         7                          011100(28)
   low                         8                          n/a
   low                         9                          n/a
   low                         10                         010000(16)
   low                         11                         n/a
```

low	12	n/a
low	13	n/a
low	14	n/a
low	15	111100(60)
high	0	010010(18)
high	1	n/a
high	2	n/a
high	3	n/a
high	4	n/a
high	5	n/a
high	6	n/a
high	7	n/a
high	8	n/a
high	9	n/a
high	10	n/a
high	11	n/a
high	12	n/a
high	13	n/a
high	14	n/a
high	15	n/a
medium-low	0	001110(14)
medium-low	1	110000(48)
medium-low	2	010110(22)
medium-low	3	100000(32)
medium-low	4	n/a
medium-low	5	n/a
medium-low	6	n/a
medium-low	7	n/a
medium-low	8	n/a
medium-low	9	n/a
medium-low	10	n/a
medium-low	11	n/a
medium-low	12	n/a
medium-low	13	n/a
medium-low	14	n/a
medium-low	15	n/a
medium-high	0	000001(1)
medium-high	1	n/a
medium-high	2	n/a
medium-high	3	n/a
medium-high	4	001001(9)
medium-high	5	n/a
medium-high	6	n/a

```
   medium-high              7                            n/a
   medium-high              8                            n/a
   medium-high              9                            n/a
   medium-high              10                           n/a
   medium-high              11                           n/a
   medium-high              12                           n/a
   medium-high              13                           n/a
   medium-high              14                           n/a
   medium-high              15                           n/a

======================================================================



Re-Write name: RE6_1 Traffic-class: dscpv6
======================================================================
loss priority            Forwarding-class            re-write prio
======================================================================
   low                      0                            001100(12)
   low                      1                            n/a
   low                      2                            n/a
   low                      3                            101111(47)
   low                      4                            001001(9)
   low                      5                            010100(20)
   low                      6                            111010(58)
   low                      7                            011100(28)
   low                      8                            n/a
   low                      9                            n/a
   low                      10                           010000(16)
   low                      11                           n/a
   low                      12                           n/a
   low                      13                           n/a
   low                      14                           n/a
   low                      15                           111100(60)
   high                     0                            010010(18)
   high                     1                            n/a
   high                     2                            n/a
   high                     3                            n/a
   high                     4                            n/a
   high                     5                            n/a
   high                     6                            n/a
   high                     7                            n/a
   high                     8                            n/a
   high                     9                            n/a
```

```
    high                       10                          n/a
    high                       11                          n/a
    high                       12                          n/a
    high                       13                          n/a
    high                       14                          n/a
    high                       15                          n/a
    medium-low                 0                           001110(14)
    medium-low                 1                           110000(48)
    medium-low                 2                           010110(22)
    medium-low                 3                           100000(32)
    medium-low                 4                           n/a
    medium-low                 5                           n/a
    medium-low                 6                           n/a
    medium-low                 7                           n/a
    medium-low                 8                           n/a
    medium-low                 9                           n/a
    medium-low                 10                          n/a
    medium-low                 11                          n/a
    medium-low                 12                          n/a
    medium-low                 13                          n/a
    medium-low                 14                          n/a
    medium-low                 15                          n/a
    medium-high                0                           000001(1)
    medium-high                1                           n/a
    medium-high                2                           n/a
    medium-high                3                           n/a
    medium-high                4                           n/a
    medium-high                5                           n/a
    medium-high                6                           n/a
    medium-high                7                           n/a
    medium-high                8                           n/a
    medium-high                9                           n/a
    medium-high                10                          n/a
    medium-high                11                          n/a
    medium-high                12                          n/a
    medium-high                13                          n/a
    medium-high                14                          n/a
    medium-high                15                          n/a

======================================================================
```

Verify ACLs for MF Classifier

```
bash-5.1# acl --list-filters --family inet
=======================================
Filter: MCFF_3
=======================================
Term: 1
-----
Priority:                256
Src ports:            [1005 - 1005]
POLICER Id:                      0
Action:                 policer (V4MCFF_31trTCM_2)
Action:                 fc (af11)
=======================================
=======================================
Filter: MCFF_3
=======================================
Term: 2
-----
Priority:                255
ICMP Type:       8
POLICER Id:                      1
Action:                 policer (V4MCFF_32trTCM_2)
Action:                 fc (af31)
=======================================
=======================================
Filter: MCFF_3
=======================================
Term: 3
-----
Priority:                254
Action:                 accept (n/a)
=======================================
bash-5.1#
bash-5.1# acl --list-filters --family inet6
=======================================
Filter: MCFF6_3
=======================================
Term: 1
-----
Priority:                256
Dest IP:                ::/0
```

```
Src IP:   ::/0
Src ports:            [1005 - 1005]
POLICER Id:                      2
Action:                 policer (V6MCFF6_31trTCM_2)
Action:                 fc (af11)
=======================================
=======================================
Filter: MCFF6_3
=======================================
Term: 2
-----
Priority:                255
Dest IP:                ::/0
Src IP:   ::/0
Dst ports:           [2152 - 2152]
POLICER Id:                      3
Action:                 policer (V6MCFF6_32trTCM_2)
Action:                 fc (nc2)
=======================================
=======================================
Filter: MCFF6_3
=======================================
Term: 3
-----
Priority:                254
Dest IP:                ::/0
Src IP:   ::/0
Action:                 accept (n/a)
=======================================
bash-5.1#
```

NOTE: Each term has its own policer instance, for example Term 1 for family inet has policer instance `V4MCFF_31trTCM_2` and Term 2 has policer instance `V4MCFF_32trTCM_2` for the same policer `srTCM_2`.

Verify Policer Statistics

```
bash-5.1# qosdpdk --get policer --name V4MCFF_32trTCM_2
=====================================
Policer parameters:
```

```
=====================================
                policer name: V4MCFF_32trTCM_2
                policer id: 1
                policer type: trTCM
                policer loss_priority: high
                policer tcm_action: tcm_discard
                policer color type: color_aware
                policer_cir: 6250000
                policer_pir: 8750000
                policer_cbs: 10000
                policer_pbs: 10000
                rx_pkts: 1
             rx_bytes: 54
             drop_pkts: 1
             drop_bytes: 54

=====================================
```

```
bash-5.1# acl --family inet --filter FF_1 --action V4FF_11srTCM_1
inet filter "FF_1": Policer "V4FF_11srTCM_1"
                        Rx Packets: 1 Rx Bytes: 54
               Dropped Packets: 1 Dropped Bytes: 54
```

Verify Scheduler Statistics

```
bash-5.1# qosdpdk --get sch --name VOICE-SCHED-MAP
Fetch Qos FC info
Scheduler name: VOICE-SCHED-MAP Scheduler  Index: 0
================================================================
 priority      forwarding      shaping      drop
               class           rate         profiles
                                            low,high,med
================================================================
strict-high(0)   none            NA             NA,NA,NA
high(1)          af12(65)        50.000 Mbps    0,0,0
medium-high(2)   none            NA             NA,NA,NA
medium-low(3)    none            NA             NA,NA,NA
low-latency(4)   none            NA             NA,NA,NA
low-high(5)      none            NA             NA,NA,NA
low-medium(6)    none            NA             NA,NA,NA
```

```
low(7)           af11(64)        50.000 Mbps     1,1,0
```

Verify dropper statistics:

```
bash-5.1# qosdpdk --get dropper --name dp1
Fetch Qos FC info
Drop Profiles
============================================================
Idx  Name        Min          Max          Mark Probability
                 Threshold    Threshold    Denominator
============================================================
0    dp1          25           75           90
```

Verify interface rate statistics:

```
bash-5.1# scheduler --rate

Interface rate statistics
-------------------------
Interface Vif Scheduler    Sched-core-Rx    Sched-Port-Rx   Sched-Port-Tx
         ID   core       Errors   Packets   Errors   Packets   Errors   Packets

eno2        2    4          0        123123   1003   123123       0     123123
```

Verify scheduler port statistics:

```
bash-5.1# scheduler --port 2
Scheduler port stats for vif 2, lcore id 4
-------------------------------------------------------------------------------------------------
-----------------------
Prio    Pkts                             Bytes
Pkts                              Bytes                        Avg     Wred
        OK                               OK
Dropped                           Dropped                      Qlen    Drops
-------------------------------------------------------------------------------------------------
-----------------------
[00]    23302064                 23768105280            4325778
4412293560              0       0
[01]    23301994                 23768033880            4325848
4412364960              0       0
```

```
[02]    23301921                23767959420             4325921
4412439420              0       0
[03]    23301913                23767951260             4325930
4412448600              0       0
[04]    27627842                28180398840             0
0                       0       0
[05]    27627842                28180398840             0
0                       0       0
[06]    26805889                27342006780             821954
838393080               0       266904
[07]    5195092308              5298994154160           1084208
1105892160              32      0
-------------------------------------------------------------------------------------------------
-----------------------
```

Verify scheduler port rate statistics:

```
bash-5.1# scheduler --portrate 2

Scheduler port stats for vif0/2
-------------------------------
Scheduler port stats for vif 2, lcore id 4
--------------------------------------------------------------------
Prio    Pkts     Bytes       Pkts       Bytes       Avg       Wred
        OK       OK          Dropped    Dropped     Qlen      Drops
--------------------------------------------------------------------
[00]    68195    102839272   0          0           0         0
[01]    136391   205678545   0          0           0         0
[02]    68196    102840779   0          0           0         0
[03]    68197    102842285   0          0           0         0
[04]    68196    102840779   0          0           0         0
[05]    68196    102840779   0          0           0         0
[06]    68196    102840779   0          0           0         0
[07]    136390   205677038   0          0           0         0

---------------------------------------------------------------------
```

Clear scheduler port statistics:

```
bash-5.1# scheduler --port 2 --clear
Counters cleared for scheduler port 2
```

Verify CoS Features Applied on an Interface

```
bash-5.1# vif --get 2
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled
       LsDp=Link down in DP only, Ccc=CCC Enabled, HwTs=Hardware support for Timestamp

vif0/2      PCI: 0000:ca:01.2 (Speed 10000, Duplex 1) NH: 8 MTU: 9216
            Type:Physical HWaddr:48:5a:0d:6f:b2:1a IPaddr:10.0.1.1
            IP6addr:1001::1
            DDP: ON SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3Vof QOS:0 Ref:17
            RX port   packets:37865806023 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:ca:01.2  Status: UP  Driver: net_iavf
            Qos classifier: dscp:BA_1 dscpv6:BA6_1
                Rewrite:    dscp:RE_1 dscpv6:RE6_1
                Scheduler:  VOICE-SCHED-MAP
            RX packets:3160058  bytes:3910471152 errors:0
            TX packets:1224847957  bytes:186064373590 errors:0
            Drops:13116
            Scheduler Enqueue   packets:1218217596 errors:0
            Scheduler Dequeue   packets:1218217596 errors:0
            TX queue  packets:1218219665 errors:6627728
            TX port   packets:1218218121 errors:0
            inet acl MCFF_1
            inet6 acl MCFF6_1
```

Two-Way Active Measurement Protocol (TWAMP)

SUMMARY

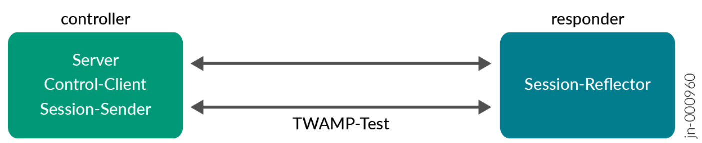
The Cloud-Native Router supports Two-Way Active Management Protocol (TWAMP) for network performance measurement and monitoring in 5G transport networks. It supports managed and light TWAMP.

IN THIS SECTION

The Two-Way Active Management Protocol (TWAMP), described in RFC 5357, is a network performance measurement and monitoring service used for active performance monitoring of 5G transport networks. TWAMP is an extension of the One-Way Active Management Protocol (OWAMP) providing two-way or round-trip measurements instead of unidirectional capabilities. Two-way measurements do not require local and remote clock synchronization. The remote host support can be limited to a simple echo function. TWAMP defines an open protocol for measuring two-way or round-trip metrics with greater accuracy than other methods by using time-stamps, while accounting for processing delays. Please review Understanding Two-Way Active Measurement Protocol topic for more details.

Juniper Cloud-Native Router supports two flavors of TWAMP implementation:

- Managed TWAMP—A TCP control connection is established between control client and responder server for exchanging test session information. Measurement and monitoring tests run between session-sender and session-reflector.
- Light TWAMP—No control connection is established between the control client and responder server. The session-sender directly runs measurement and monitoring tests with the session-reflector. The session-reflector has no knowledge of the session state.



Cloud-Native Router supports timestamping the TWAMP test packets by either the cRPD or vRouter. The cRPD timestamps are enabled by default. Since the Round Trip Time (RTT) for cRPD timestamps has higher jitter due to latency in the host network stack, vRouter timestamps may be preferred. The vRouter uses hardware timestamping if the underlying NIC supports it, else it defaults to the kernel system clock.

NOTE: vRouter timestamping is supported for TWAMP light mode only.

Configuration

To enable vRouter timestamping, you must enable the `twampPort` configuration in the Cloud-Native Router helm chart at the time of installation. The vRouter listens to TWAMP test messages on the configured `twampPort` and overwrites timestamps in TWAMP test packets. Please review the *Customizing Cloud-Native Router Helm Chart* for more details. When not configured, cRPD timestamps the TWAMP test packets.

You can configure the TWAMP server and client with minimum configuration. There are additional configuration parameters with default values which may be modified as per your requirement. Please review edit services rpm twamp command for more information on each configuration option. The default values for the options is provided in the below tables:

Table 12: TWAMP Client Default Values

Option	Default value
control-type (light | managed)	managed
destination-port (862 - 65535)	862
history-size (0 - 512)	50
moving-average-size (0 - 512)	0
persistent-results (enable | disable)	disable
target-address	An IPv4 address. This field is mandatory for managed control-type. The configuration commit fails if configured for light control-type.
tcp-keepcnt (1 - 50)	6

Table 12: TWAMP Client Default Values *(Continued)*

Option	Default value
tcp-keepidle (1 - 600 seconds)	120
tcp-keepintvl (1 - 600 seconds)	5
test-count (0 - 4294967290)	0
test-interval (1 - 255)	1
test-session (name)	Mandatory
data-size (60 - 1400)	60
destination-port (862 - 65535)	862
dscp-code-points	000000
probe-count (1 - 4294967290)	1
probe-interval (1 - 255)	1

Table 13: TWAMP Server Default Values

Options	Values
port (862 - 65535) [*light*]	862
max-connection-duration (0 - 120 hours)	24
maximum-connections (0 - 1000)	64
maximum-connections-per-client (1 - 500)	64

Table 13: TWAMP Server Default Values *(Continued)*

Options	Values
maximum-sessions (1 - 2048)	64
maximum-sessions-per-connection (1 - 1024)	64
port (1 - 65535) [*server*]	862
port (1 - 65535) [*routing-instance-list*]	862
server-inactivity-timeout (0 - 30 minutes)	15
tcp-keepcnt (1 - 50)	6
tcp-keepidle (1 - 600 seconds)	120
tcp-keepintvl (1 - 600 seconds)	5

Sample TWAMP client and server configurations for managed or TWAMP light are provided below. Use the *configlet resource* to configure cRPD:

TWAMP Client/Server Configuration (Managed, Minimum Configuration)

Client Configuration

```
set services rpm twamp client control-connection myTcManaged1 target-address 1.1.1.29
set services rpm twamp client control-connection myTcManaged1 test-session myTs1 target-address
21.21.21.29
```

Server Configuration

```
set services rpm twamp server client-list myClients address 21.21.21.0/24
```

TWAMP Client/Server Configuration (Managed, Optional Configuration)

Client Configuration

```
set services rpm twamp client control-connection myTcManaged1 control-type managed
set services rpm twamp client control-connection myTcManaged1 destination-interface ens2f0
set services rpm twamp client control-connection myTcManaged1 destination-port 10000
set services rpm twamp client control-connection myTcManaged1 history-size 50
set services rpm twamp client control-connection myTcManaged1 moving-average-size 50
set services rpm twamp client control-connection myTcManaged1 persistent-results
set services rpm twamp client control-connection myTcManaged1 routing-instance routing-instance
set services rpm twamp client control-connection myTcManaged1 source-address 2.2.2.29
set services rpm twamp client control-connection myTcManaged1 target-address 21.21.21.29
set services rpm twamp client control-connection myTcManaged1 tcp-keepcnt 10
set services rpm twamp client control-connection myTcManaged1 tcp-keepidle 60
set services rpm twamp client control-connection myTcManaged1 tcp-keepintvl 600
set services rpm twamp client control-connection myTcManaged1 test-count 3
set services rpm twamp client control-connection myTcManaged1 test-interval 10
set services rpm twamp client control-connection myTcManaged1 test-session test1 data-fill-with-
zeros
set services rpm twamp client control-connection myTcManaged1 test-session test1 data-size 100
set services rpm twamp client control-connection myTcManaged1 test-session test1 destination-
port 65000
set services rpm twamp client control-connection myTcManaged1 test-session test1 dscp-code-
points 000001
set services rpm twamp client control-connection myTcManaged1 test-session test1 probe-count 10
set services rpm twamp client control-connection myTcManaged1 test-session test1 probe-interval 1
set services rpm twamp client control-connection myTcManaged1 test-session test1 source-address
21.21.21.30
set services rpm twamp client control-connection myTcManaged1 test-session test1 target-address
21.21.21.29
set services rpm twamp client control-connection myTcManaged1 test-session test1 ttl 5
```

Server Configuration

```
set services rpm twamp server authentication-mode none
set services rpm twamp server client-list 192.168.11.0/24
set services rpm twamp server max-connection-duration 1
set services rpm twamp server maximum-connections 20
set services rpm twamp server maximum-connections-per-client 20
set services rpm twamp server maximum-sessions 30
```

```
set services rpm twamp server maximum-sessions-per-connection 30
set services rpm twamp server port 10000
set services rpm twamp server routing-instance-list <routing-instance> <port>
set services rpm twamp server server-inactivity-timeout 10
set services rpm twamp server tcp-keepcnt 10
set services rpm twamp server tcp-keepidle 60
set services rpm twamp server tcp-keepintvl 600
```

TWAMP Client/Server Configuration (Light, Minimum Configuration)

Client Configuration

```
set services rpm twamp client control-connection myTcLight1 control-type light
set services rpm twamp client control-connection myTcLight1 test-session myTs1 target-address
21.21.21.29
```

Server Configuration

```
set services rpm twamp server light
```

TWAMP Client/Server Configuration (Light, Optional Configuration)

Client Configuration

```
set services rpm twamp client control-connection myTcLight1 control-type light
set services rpm twamp client control-connection myTcLight1 test-session test1 data-fill-with-
zeros
set services rpm twamp client control-connection myTcLight1 test-session test1 data-size 100
set services rpm twamp client control-connection myTcLight1 test-session test1 destination-port
65000
set services rpm twamp client control-connection myTcLight1 test-session test1 dscp-code-points
000001
set services rpm twamp client control-connection myTcLight1 test-session test1 probe-count 10
set services rpm twamp client control-connection myTcLight1 test-session test1 probe-interval 1
set services rpm twamp client control-connection myTcLight1 test-session test1 source-address
21.21.21.30
set services rpm twamp client control-connection myTcLight1 test-session test1 target-address
```

```
21.21.21.29
set services rpm twamp client control-connection myTcLight1 test-session test1 ttl 5
```

Server Configuration

```
set services rpm twamp server control-type light
```

NOTE: By default the client control connection `test-count` is set to zero. In this case the TWAMP test automatically starts after the configuration is committed and continues to run until the configuration is deleted. If `test-count` is configured to a non-zero value, the TWAMP test must be started or stopped using below commands:

```
user@host> request services rpm twamp start client control-client-name
user@host> request services rpm twamp stop client control-client-name
```

Verification

- You can use the `show services rpm twamp client probe-results` command to verify the TWAMP probe results on "cRPD" on page 329 shell:

```
user@host> show services rpm twamp client probe-results
    Owner: myTcManaged1, Test: myTs1
    server-address: 1.1.1.29, server-port: 862, Client address: 21.21.21.30, Client port: 
35109
    TWAMP-Server-Status: Connected, Number-Of-Retries-With-TWAMP-Server: 222
    Reflector address: 21.21.21.29, Reflector port: 10029, Sender address: 21.21.21.30, 
sender-port: 10029
    Test size: 1 probes
    Probe results:
      Response received
      Probe sent time: Thu Jun 13 06:34:14 2024
      Probe rcvd/timeout time: Thu Jun 13 06:34:14 2024
      Rtt: 968 usec, Egress jitter: 63 usec, Ingress jitter: -22 usec, Round trip jitter: 28 
usec
      Egress interarrival jitter: 40 usec, Ingress interarrival jitter: 9 usec, Round trip 
interarrival jitter: 32 usec
    Results over current test:
```

```
      Probes sent: 1, Probes received: 1, Loss percentage: 0.000000
      Measurement: Round trip time
        Samples: 1, Minimum: 968 usec, Maximum: 968 usec, Average: 968 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 968 usec
      Measurement: Positive egress jitter
        Samples: 1, Minimum: 63 usec, Maximum: 63 usec, Average: 63 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 63 usec
      Measurement: Negative ingress jitter
        Samples: 1, Minimum: 22 usec, Maximum: 22 usec, Average: 22 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 22 usec
      Measurement: Positive round trip jitter
        Samples: 1, Minimum: 28 usec, Maximum: 28 usec, Average: 28 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 28 usec
    Results over last test:
      Probes sent: 1, Probes received: 1, Loss percentage: 0.000000
      Test completed on Thu Jun 13 06:34:14 2024
      Measurement: Round trip time
        Samples: 1, Minimum: 968 usec, Maximum: 968 usec, Average: 968 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 968 usec
      Measurement: Positive egress jitter
        Samples: 1, Minimum: 63 usec, Maximum: 63 usec, Average: 63 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 63 usec
      Measurement: Negative ingress jitter
        Samples: 1, Minimum: 22 usec, Maximum: 22 usec, Average: 22 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 22 usec
      Measurement: Positive round trip jitter
        Samples: 1, Minimum: 28 usec, Maximum: 28 usec, Average: 28 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 28 usec
    Results over all tests:
      Probes sent: 5, Probes received: 5, Loss percentage: 0.000000
      Measurement: Round trip time
        Samples: 5, Minimum: 892 usec, Maximum: 1186 usec, Average: 992 usec, Peak to peak: 
294 usec, Stddev: 102 usec, Sum: 4958 usec
      Measurement: Positive egress jitter
        Samples: 3, Minimum: 63 usec, Maximum: 229 usec, Average: 125 usec, Peak to peak: 166 
usec, Stddev: 74 usec, Sum: 375 usec
      Measurement: Negative egress jitter
        Samples: 1, Minimum: 354 usec, Maximum: 354 usec, Average: 354 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 354 usec
      Measurement: Positive ingress jitter
        Samples: 1, Minimum: 60 usec, Maximum: 60 usec, Average: 60 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 60 usec
      Measurement: Negative ingress jitter
```

```
        Samples: 3, Minimum: 22 usec, Maximum: 48 usec, Average: 33 usec, Peak to peak: 26 
usec, Stddev: 11 usec, Sum: 98 usec
      Measurement: Positive round trip jitter
        Samples: 3, Minimum: 28 usec, Maximum: 203 usec, Average: 98 usec, Peak to peak: 175 
usec, Stddev: 75 usec, Sum: 295 usec
      Measurement: Negative round trip jitter
        Samples: 1, Minimum: 298 usec, Maximum: 298 usec, Average: 298 usec, Peak to peak: 0 
usec, Stddev: 0 usec, Sum: 298 usec
```

- Additional show commands include:

```
show services rpm twamp client
show services rpm twamp client connection connection-name
show services rpm twamp client history-results
show services rpm twamp client history-results brief
show services rpm twamp client history-results control-connection control-connection
show services rpm twamp client history-results detail
show services rpm twamp client history-results detail control-connection control-connection
show services rpm twamp client history-results detail control-connection control-connection 
test-session test-session
show services rpm twamp client history-results detail since YYYY-MM-DD.HH:MM:SS
show services rpm twamp client probe-results
show services rpm twamp client probe-results control-connection control-connection
show services rpm twamp client probe-results control-connection control-connection test-
session test-session
show services rpm twamp client session
show services rpm twamp client session control-connection control-connection test-session 
test-session
show services rpm twamp server
show services rpm twamp server connection connection-id
show services rpm twamp server session session-id
```

- If vRouter timestamping is enabled, you can verify hardware timestamping support for a fabric interface by executing the following command on the "vRouter CLI" on page 331:

```
bash-5.1# dpdkinfo -p 1

********************* Infos for port 0  *********************
MAC address: B4:96:91:DA:D4:F8
Device name: 0000:af:00.0
Driver name: net_ice
```

```
..
..
..
Max segment number per packet: 0
Max segment number per MTU/TSO: 0
Hardware timestamping support: Rx:Yes, Tx:Yes
Device capabilities: 0x0( )
```

- If the underlying NIC supports hardware timestamping and vRouter timestamping is enabled, you can verify hardware timestamp support for the fabric interface by executing the following command on the "vRouter CLI" on page 331:

```
bash-5.1# vif --get 1

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload,
Mon=Interface is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled
       LsDp=Link down in DP only, Ccc=CCC Enabled, HwTs=Hardware support for Timestamp

vif0/1      PCI: 0000:af:00.0 (Speed 10000, Duplex 1) NH: 6 MTU: 9000
            Type:Physical HWaddr:b4:96:91:da:d4:f8 IPaddr:110.110.110.22
            IP6addr:110:110::110:22
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:TcL3VofHwTs QOS:0 Ref:14
            RX port   packets:93509698 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:af:00.0  Status: UP  Driver: net_ice
            RX packets:93509698  bytes:8602892343 errors:0
            TX packets:124162093  bytes:11422912735 errors:0
            Drops:324
            TX port   packets:124162084 errors:0
```

The TWAMP counters are incremented for the corresponding tap interface as well:

```
bash-5.1# vif --get 2

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload,
Mon=Interface is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled
       LsDp=Link down in DP only, Ccc=CCC Enabled, HwTs=Hardware support for Timestamp

vif0/2      PMD: enp175s0f0 NH: 9 MTU: 9000
            Type:Host HWaddr:b4:96:91:da:d4:f8 IPaddr:110.110.110.22
            IP6addr:110:110::110:22
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:11 TxXVif:1
            RX device packets:182  bytes:17063 errors:0
            RX queue  packets:182 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Twamp RX  packets:84 errors:0
            Twamp TX  packets:84 errors:0
            RX packets:182  bytes:17063 errors:0
            TX packets:174  bytes:16189 errors:0
            Drops:0
            TX queue  packets:174 errors:0
            TX device packets:174  bytes:16189 errors:0
```

Segment Routing

SUMMARY

The Juniper Cloud-Native Router provides support for Segment Routing (SR-MPLS and SRv6). Read this topic to understand the supported features.

IN THIS SECTION

Segment routing (SR) is a modern variant of source routing that simplifies the network by removing network state information from intermediate routers and instead adds path state information into packet headers in the forwarding plane. When a packet arrives at an SR ingress node, the ingress node subjects the packet to policy. The policy associates the packet with an SR path to its destination. The SR path is an ordered list of segments that connects an SR ingress node to an SR egress node. This SR path can be engineered to satisfy any number of constraints for example, link bandwidth, minimum path latency and more.

Segment Routing can leverage either MPLS or IPv6 in the forwarding plane. When Segment Routing uses the MPLS forwarding plane, it is referred to as SR-MPLS. SR-MPLS supports both an IPv4 and IPv6 underlay. When Segment Routing leverages an IPv6 forwarding plane, it is called SRv6. Review the SR-MPLS Day One Book and SRv6 Day One Book for more details.

Segment Routing can be used as a transport tunneling technology for interconnecting data centers for the next-generation Network Function Virtualization (NFV) based telco cloud, 5G, cloud WAN and content distribution networks (CDNs). It can provide multiple benefits in the following use cases:

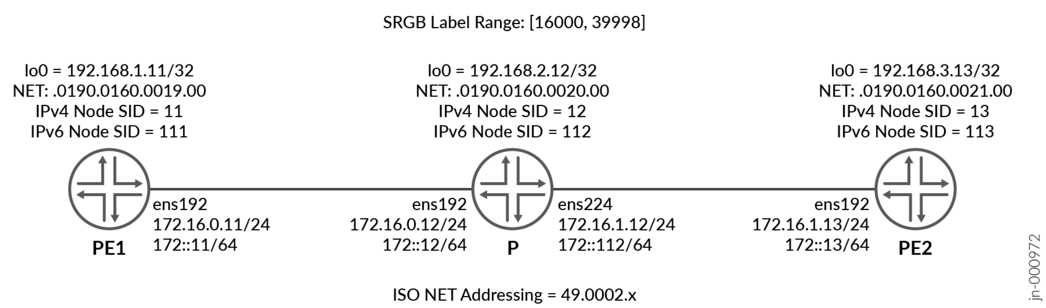
- Traffic Engineering— Segment Routing provides efficient and dynamic steering of traffic based on networking conditions to enable load balancing, congestion avoidance and better utilization of bandwidth. This results in improved network performance and user experience.
- Network Slicing— Segment Routing creates virtualized network slices by enabling different services or tenants to coexist on the same physical infrastructure. Each network slice can have its own forwarding policies and resources, providing enhanced isolation and flexibility for diverse applications.
- Service Function Chaining— Segment Routing enables flexible and dynamic service function chaining, where packets traverse a sequence of network services such as firewalls, load-balancing and more. It simplifies service deployments and enables on-demand service chaining by defining specific paths for packet flow.
- Mobile Networks— Segment Routing optimizes traffic routing and handover procedures for a mobile network by enabling efficient path selection and reduced latency.

SR-MPLS

Cloud-Native Router supports SR-MPLS for both IPv4 and IPv6 underlay. The cloud-native router can participate as a sending, receiving or transit router in SR-MPLS networks. It supports SR-MPLS implementation with or without penultimate hop popping (PHP), with and without overlay ECMP, and Explicit/Implicit NULL. Segment Routing Flexible Algorithm (Flex-Algo) is supported from Cloud-Native Router Release 24.1 onwards. Review Segment Routing (IS-IS) for more information.

Configuring SR-MPLS on JCNR

Consider the following topology:



We want to segment routing in this network. PE1 is an ingress Cloud-Native Router node, P is a transit Cloud-Native Router node while PE2 is an egress Cloud-Native Router node. We will configure a common segment routing global block (SRGB) range for all nodes. All nodes are assigned a segment ID (node SID) for both IPv4 and IPv6.

NOTE: Use the *configlet resource* to configure the cRPD pods.

Cloud-Native Router Ingress (PE1) Configuration

Configure the Cloud-Native Router Ingress (PE1) router with the following configuration:

1. Configure the interfaces:

```
set interfaces ens192 unit 0 family inet address 172.16.0.11/24
set interfaces ens192 unit 0 family inet6 address 172::11/64
set interfaces ens192 unit 0 family iso
set interfaces ens192 unit 0 family mpls
set interfaces lo0 unit 0 family inet address 192.168.1.11/32
set interfaces lo0 unit 0 family inet6 address 192::11/128
```

```
set interfaces lo0 unit 0 family iso address 49.0002.0190.0160.0019.00
set interfaces lo0 unit 0 family mpls
```

2. Configure IS-IS:

```
set protocols isis interface ens192
set protocols isis interface lo0.0
set protocols isis level 1 disable
```

3. Configure routing options:

```
set routing-options route-distinguisher-id 192.168.1.11
set routing-options router-id 192.168.1.11
set routing-options autonomous-system 64512
```

4. Configure MPLS protocol on the interfaces:

```
set protocols mpls ipv6-tunneling
set protocols mpls interface ens192
set protocols mpls interface lo0.0
```

5. Configure BGP neighbors:

```
set protocols bgp group sr_mpls type internal
set protocols bgp group sr_mpls multihop
set protocols bgp group sr_mpls local-address 192.168.1.11
set protocols bgp group sr_mpls family inet-vpn unicast
set protocols bgp group sr_mpls family inet6-vpn unicast
set protocols bgp group sr_mpls local-as 64512
set protocols bgp group sr_mpls neighbor 192.168.3.13
```

6. Configure the start label and index range of SRGB:

```
set protocols isis source-packet-routing srgb start-label 16000
set protocols isis source-packet-routing srgb index-range 23999
```

7. Configure the IPv4 and IPv6 node segment ID:

```
set protocols isis source-packet-routing node-segment ipv4-index 11
set protocols isis source-packet-routing node-segment ipv6-index 111
```

8. Optionally, configure the explicit null label:

```
set protocols isis source-packet-routing explicit-null
```

Cloud-Native Router Transit (P) Configuration

Configure the Cloud-Native Router Transit (P) router with the following configuration:

1. Configure the interfaces:

```
set interfaces ens192 unit 0 family inet address 172.16.0.12/24
set interfaces ens192 unit 0 family inet6 address 172::12/64
set interfaces ens192 unit 0 family iso
set interfaces ens192 unit 0 family mpls
set interfaces ens224 unit 0 family inet address 172.16.1.12/24
set interfaces ens224 unit 0 family inet6 address 172::112/64
set interfaces ens224 unit 0 family iso
set interfaces ens224 unit 0 family mpls
set interfaces lo0 unit 0 family inet address 192.168.2.12/32
set interfaces lo0 unit 0 family inet6 address 192::12/128
set interfaces lo0 unit 0 family iso address 49.0002.0190.0160.0020.00
set interfaces lo0 unit 0 family mpls
```

2. Configure IS-IS:

```
set protocols isis interface ens192
set protocols isis interface ens224
set protocols isis interface lo0.0
set protocols isis level 1 disable
```

3. Configure routing options:

```
set routing-options route-distinguisher-id 192.168.2.12
set routing-options router-id 192.168.2.22
set routing-options autonomous-system 64512
```

4. Configure MPLS protocol on the interfaces:

```
set protocols mpls interface ens192
set protocols mpls interface lo0.0
set protocols mpls interface ens224
```

5. Configure the start label and index range of SRGB:

```
set protocols isis source-packet-routing srgb start-label 16000
set protocols isis source-packet-routing srgb index-range 23999
```

6. Configure the IPv4 and IPv6 node segment ID:

```
set protocols isis source-packet-routing node-segment ipv4-index 11
set protocols isis source-packet-routing node-segment ipv6-index 111
```

Cloud-Native Router Egress (PE2) Configuration

Configure the Cloud-Native Router Egress (PE2) router with the following configuration:

1. Configure the interfaces:

```
set interfaces ens192 unit 0 family inet address 172.16.1.13/24
set interfaces ens192 unit 0 family inet6 address 172::13/64
set interfaces ens192 unit 0 family iso
set interfaces ens192 unit 0 family mpls
set interfaces lo0 unit 0 family inet address 192.168.3.13/32
set interfaces lo0 unit 0 family inet6 address 192::13/128
set interfaces lo0 unit 0 family iso address 49.0002.0190.0160.0021.00
set interfaces lo0 unit 0 family mpls
```

2. Configure IS-IS:

```
set protocols isis interface ens192
set protocols isis interface lo0.0
set protocols isis level 1 disable
```

3. Configure routing options:

```
set routing-options route-distinguisher-id 192.168.3.13
set routing-options router-id 192.168.3.13
set routing-options autonomous-system 64512
```

4. Configure MPLS protocol on the interfaces:

```
set protocols mpls ipv6-tunneling
set protocols mpls interface ens192
set protocols mpls interface lo0.0
```

5. Configure BGP neighbors:

```
set protocols bgp group sr_mpls type internal
set protocols bgp group sr_mpls multihop
set protocols bgp group sr_mpls local-address 192.168.3.13
set protocols bgp group sr_mpls family inet-vpn unicast
set protocols bgp group sr_mpls family inet6-vpn unicast
set protocols bgp group sr_mpls local-as 64512
set protocols bgp group sr_mpls neighbor 192.168.1.11
```

6. Configure the start label and index range of SRGB:

```
set protocols isis source-packet-routing srgb start-label 16000
set protocols isis source-packet-routing srgb index-range 23999
```

7. Configure the IPv4 and IPv6 node segment ID:

```
set protocols isis source-packet-routing node-segment ipv4-index 13
set protocols isis source-packet-routing node-segment ipv6-index 113
```

8. Optionally, configure the explicit null label:

```
set protocols isis source-packet-routing explicit-null
```

Verify SR-MPLS Configuration on JCNR

The following commands can be used to verify the SRv6 configuration on "cRPD" on page 329:

```
user@pe1> show isis database detail
IS-IS level 1 link-state database:

IS-IS level 2 link-state database:

pe1.00-00 Sequence: 0x125, Checksum: 0xce5c, Lifetime: 590 secs
  IPV4 Index: 11, IPV6 Index: 111
  Node Segment Blocks Advertised:
    Start Index : 0, Size : 23999, Label-Range: [ 16000, 39998 ]
   IS neighbor: node2.02                      Metric:       10
     LAN IPv4 Adj-SID:      16, Weight:   0, Neighbor: node3, Flags: --VL--
     LAN IPv6 Adj-SID:      17, Weight:   0, Neighbor: node3, Flags: F-VL--
   IP prefix: 172.16.0.0/24                   Metric:       10 Internal Up
   IP prefix: 192.168.1.11/32                 Metric:        0 Internal Up
   V6 prefix: 172::/64                        Metric:       10 Internal Up
   V6 prefix: 192::11/128                     Metric:        0 Internal Up
   V6 prefix: fe80::50c8:9dff:fee2:4655/128   Metric:        0 Internal Up

pe1.02-00 Sequence: 0x11f, Checksum: 0x4305, Lifetime: 468 secs
   IS neighbor: node2.00                      Metric:        0
   IS neighbor: node3.00                      Metric:        0

p.00-00 Sequence: 0x83, Checksum: 0xcd5e, Lifetime: 506 secs
  IPV4 Index: 12, IPV6 Index: 112
  Node Segment Blocks Advertised:
    Start Index : 0, Size : 23999, Label-Range: [ 16000, 39998 ]
   IS neighbor: node2.02                      Metric:       10
```

```
      LAN IPv4 Adj-SID:      16, Weight:   0, Neighbor: node2, Flags: --VL--
      LAN IPv6 Adj-SID:      17, Weight:   0, Neighbor: node2, Flags: F-VL--
    IS neighbor: node3.02                     Metric:       10
      LAN IPv4 Adj-SID:      20, Weight:   0, Neighbor: node4, Flags: --VL--
      LAN IPv6 Adj-SID:      21, Weight:   0, Neighbor: node4, Flags: F-VL--
    IP prefix: 172.16.0.0/24                  Metric:       10 Internal Up
    IP prefix: 172.16.1.0/24                  Metric:       10 Internal Up
    IP prefix: 192.168.2.12/32                Metric:        0 Internal Up
    V6 prefix: 172::/64                       Metric:       10 Internal Up
    V6 prefix: 192::12/128                    Metric:        0 Internal Up
    V6 prefix: fe80::50e4:70ff:fe46:76dd/128   Metric:        0 Internal Up

p.02-00 Sequence: 0x78, Checksum: 0xf2e5, Lifetime: 1156 secs
    IS neighbor: node3.00                     Metric:        0
    IS neighbor: node4.00                     Metric:        0

pe2.00-00 Sequence: 0x76, Checksum: 0xb5bf, Lifetime: 644 secs
  IPV4 Index: 13, IPV6 Index: 113
  Node Segment Blocks Advertised:
    Start Index : 0, Size : 23999, Label-Range: [ 16000, 39998 ]
    IS neighbor: node3.02                     Metric:       10
      LAN IPv4 Adj-SID:      16, Weight:   0, Neighbor: node3, Flags: --VL--
      LAN IPv6 Adj-SID:      17, Weight:   0, Neighbor: node3, Flags: F-VL--
    IP prefix: 172.16.1.0/24                  Metric:       10 Internal Up
    IP prefix: 192.168.3.13/32                Metric:        0 Internal Up
    V6 prefix: 172::/64                       Metric:       10 Internal Up
    V6 prefix: 192::13/128                    Metric:        0 Internal Up
    V6 prefix: fe80::3c1e:39ff:fe28:1a8a/128   Metric:        0 Internal Up
```

```
user@pe1> show route table inet.3

inet.3: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.2.12/32    *[L-ISIS/14] 1d 01:20:51, metric 10
                    >  to 172.16.0.12 via ens192
```

```
192.168.3.13/32    *[L-ISIS/14] 1d 00:59:01, metric 20
                    >  to 172.16.0.12 via ens192, Push 16013
```

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

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

0                  *[MPLS/0] 1d 02:05:44, metric 1
                       Receive
1                  *[MPLS/0] 1d 02:05:44, metric 1
                       Receive
2                  *[MPLS/0] 1d 02:05:44, metric 1
                       Receive
13                 *[MPLS/0] 1d 02:05:44, metric 1
                       Receive
16                 *[L-ISIS/14] 1d 01:12:40, metric 0
                    >  to 172.16.0.11 via ens192, Pop
16(S=0)            *[L-ISIS/14] 1d 01:12:40, metric 0
                    >  to 172.16.0.11 via ens192, Pop
17                 *[L-ISIS/14] 1d 01:12:40, metric 0
                    >  to fe80::250:56ff:fea9:5f96 via ens192, Pop
17(S=0)            *[L-ISIS/14] 1d 01:12:40, metric 0
                    >  to fe80::250:56ff:fea9:5f96 via ens192, Pop
20                 *[L-ISIS/14] 1d 01:13:06, metric 0
                    >  to 172.16.1.13 via ens224, Pop
20(S=0)            *[L-ISIS/14] 1d 01:13:06, metric 0
                    >  to 172.16.1.13 via ens224, Pop
21                 *[L-ISIS/14] 1d 01:13:06, metric 0
                    >  to fe80::250:56ff:fea9:5dc via ens224, Pop
21(S=0)            *[L-ISIS/14] 1d 01:13:06, metric 0
                    >  to fe80::250:56ff:fea9:5dc via ens224, Pop
16011              *[L-ISIS/14] 1d 01:00:38, metric 10
                    >  to 172.16.0.11 via ens192, Swap 0
16011(S=0)         *[L-ISIS/14] 1d 01:12:40, metric 10
                    >  to 172.16.0.11 via ens192, Pop
16013              *[L-ISIS/14] 1d 01:00:24, metric 10
                    >  to 172.16.1.13 via ens224, Swap 0
16013(S=0)         *[L-ISIS/14] 1d 01:12:57, metric 10
                    >  to 172.16.1.13 via ens224, Pop
16111              *[L-ISIS/14] 1d 01:00:38, metric 10
```

```
                     >  to fe80::250:56ff:fea9:5f96 via ens192, Swap 2
16111(S=0)          *[L-ISIS/14] 1d 01:12:40, metric 10
                     >  to fe80::250:56ff:fea9:5f96 via ens192, Pop
16113               *[L-ISIS/14] 1d 01:00:24, metric 10
                     >  to fe80::250:56ff:fea9:5dc via ens224, Swap 2
16113(S=0)          *[L-ISIS/14] 1d 01:12:57, metric 10
                     >  to fe80::250:56ff:fea9:5dc via ens224, Pop
```

Verify Configuration on Cloud-Native Router Forwarding Plane

Verify the traffic flow via the "vRouter" on page 331 on each PE node:

```
user@pe1# mpls --dump
MPLS Input Label Map

   Label    NextHop
-------------------
       0       26
       1       26
       2       26
      13       26
      16       27
      17       28
   16012       27
   16013       29
   16112       28
   16113       30
```

```
user@pe1# mpls --get 16013
MPLS Input Label Map

   Label    NextHop
-------------------
   16013       29
```

```
user@pe1# nh --get 29
Id:29         Type:Tunnel         Fmly: AF_MPLS  Rid:0  Ref_cnt:2          Vrf:0
              Flags:Valid, Policy, Etree Root, MPLS,
```

```
              Oif:1 Len:14 Data:00 50 56 a9 10 fa 00 50 56 a9 5f 96 88 47 Number of Transport 
Labels:1 Transport Labels:16013,
```

```
user@pe1# mpls --get 16
MPLS Input Label Map

   Label    NextHop
-------------------
      16        27
```

```
user@pe1# nh --get 27
Id:27         Type:Encap         Fmly:AF_INET/6  Rid:0  Ref_cnt:6         Vrf:0
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:1 Len:14
              Encap Data: 00 50 56 a9 10 fa 00 50 56 a9 5f 96
```

SRv6

SRv6 is a segment routing paradigm that is applied to an IPv6 underlay with a new IPv6 extension header called Segment Routing Header (SRH). SRv6 leverages existing IPv6 forwarding technology to encode network programming instructions, also known as Segment Identifiers (SIDs). In SRv6, the SIDs are represented as IPv6 addresses when compared with SR-MPLS where SIDs are encoded as SR-MPLS labels. An SRv6 SID is 128 bits and consists of the following components:

Table 14: SID Components

Component	Description
Locator	First part of an SID that identifies the address of an SRv6 node. It is a network address that provides route to its parent node and is installed in inet6.0 table by the IS-IS protocol. IS-IS routes the segment to its parent node, which performs a function that is defined in the second part of the SID. It is 64 bits in length.

Table 14: SID Components *(Continued)*

Component	Description
Function	Second part of an SID that defines the function that a node (identified by the locator) performs, such as: **End**: Endpoint function for SRv6 instantiation of a Prefix SID [RFC8402] **End.X**: Endpoint with L3 cross-connect function for SRv6 instantiation of an Adjacent SID [RFC8402] **End.DT4**: Endpoint with decapsulation and specific IPv4 table lookup function for SRv6 instantiation of Global or IPv4 L3VPN (transport IPv4 services over SRv6 underlay) **End.DT6**: Endpoint with decapsulation and specific IPv6 table lookup function for SRv6 instantiation of Global or IPv6 L3VPN (Transport IPv6 services over SRv6 underlay) **End.DT46**: Endpoint with decapsulation and specific IP table lookup function for SRv6 instantiation of Global, IPv4 or IPv6 L3VPN (Transport both IPv4 and IPv6 services over SRv6 underlay). It is shared across IPv4 and IPv6 prefixes. The End SID behavior can be specified through flavors such as Penultimate Segment Pop (PSP), Ultimate Segment Pop (USP), and Ultimate Segment Decapsulation (USD). The Function component is 16 bits in length.
Argument	A variable length field that provides additional information about the forwarding action. Can be maximum 48 bits in length.

The SRH carries the SIDs. As the packet travels through the network, each node examines the next SID in the SRH to determine the corresponding next-hop and forwards the packet until the packet reaches its destination. If a packet is required to be guided via multiple segments in the SRv6 path, the SRv6-SIDs need to be stacked up using the SRH, to a maximum of 6 typical SRv6-SIDs. This presents additional bandwidth and processing overhead. Thus, micro-SIDs (uSIDs) are envisaged where multiple SRV6-addresses are compressed into a single IPv6 address. The 16-byte uSID is encoded as a micro-program or a container instruction that is carried either in the packet destination address or in the SRH. It has a specific structure represented by the following format:

```
BBBB:BBBB:<uSID1>:<uSID2>:<uSID3>:<uSID4>:<uSID5>:<uSID6>
```

where `BBBB:BBBB/32` represents the prefix or block assigned by an operator within an SR domain. Various prefix lengths are supported including `/16`, `/32`, `/48`, `/64` blocks. A `/32` block is most commonly used. The blocks can either be a Global Identifier Block (GIB) or a Local Identifier Block (LIB). The GIB represents a globally unique range of uSIDs allocated from a public or reserved address space specifically designated for SRv6 deployments. The LIB is assigned by a specific network node within the SR domain and applicable only within the node's local context. The `uSIDx` represents the individual 16-bit uSIDs that can either be from the GIB or LIB. The uSIDs implement the following functions:

- uN: Micro-node-SID that maps to the ultimate destination (End).
- uA: Micro-adjacency-SID that has adjacency specific behavior (End.X)
- uDT: Micro-service-SID that is domain specific information (End.DT4, End.DT6, End.DT46)

Cloud-Native Router supports the following SRv6 functionalities:

Table 15: SRv6 Supported Functionalities

Functionalities	Notes
SRv6 L3VPN with uSID	uSID Types: Global uSID, Local uSID uSID encoding in Destination Address
Block and uSID sizes	Support for /16, /32, /48, /64 blocks
Cloud-Native Router SRv6 Node Types and Micro-instructions	Ingress: SRv6 Encapsulation in Destination Address (SRH is not required) Transit: IPv6 forwarding if Ingress node is JCNR; Shift and forward if Ingress node is non-JCNR Egress: Decpasulate and execute the SID service function
SR Endpoints (Functions)	End, End.X, End.DT4, End.DT6, and End.DT46
uSID Behavior	uN, uDT
Failure Recovery	Control plane initiated failure recovery (alternative path as next hop)

Cloud-Native Router supports the following SRv6 network programming features:

Table 16: SRv6 Supported Features

Feature	Description
SRv6 uSID underlay tunnels via IS-IS	IS-IS brings up Best Effort tunnel to advertised locators and programs them in inet6.3 table.
SRv6 uSID underlay tunnels via IS-IS with ECMP paths	IS-IS brings up Best Effort tunnel to advertised locators and programs them in inet6.3 table. The tunnels can have ECMP forwarding paths.
Routes for BGP internet prefixes advertised with uN SID	BGP internet prefixes advertised with uN SIDs will resolve over corresponding locators (SRv6 uSID underlay tunnels). The underlay routes can have a single gateway or ECMP.
Multipath routes for BGP internet prefixes advertised with uN SID	Multiple Provide Edge (PE) routers could originate the same internet prefix (multihoming) that can lead to BGP multipath at the ingress PE. Each multipath route resolves over underlay SRv6 tunnels that have either a single gateway or ECMP.
Routes for L3VPN prefixes advertised with uN SID	L3VPN prefixes advertised with uN SIDs will resolve over corresponding locators (SRV6 uSID underlay tunnels). These underlay routes can have a single gateway or have ECMP.
Multipath routes for L3VPN prefixes advertised with uN SID	Multiple PEs could originate the same L3VPN prefix (multihoming) and this can lead to BGP multipath at the ingress PE. Each multipath route resolves over the underlay SRv6 tunnels that have either a single gateway or ECMP.
BGP intent routes over flex-algorithm IS-IS tunnels	Flex algorithm uSIDs can be advertised via IS-IS with SRv6 underlay tunnels created. BGP internet prefixes and L3VPN prefixes with uSIDs can resolve over the underlay tunnels. These prefixes can be non-intent (without any color-community attached to them). The prefixes are resolved over SRV6 underlay tunnels installed in inet6.3 table.

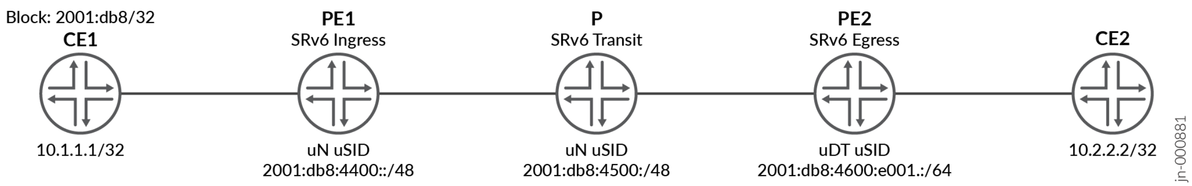
Table 16: SRv6 Supported Features *(Continued)*

Feature	Description
BGP intent routes over flex-algorithm IS-IS tunnels with fallback mechanism	Flex algorithm uSIDs can be advertised via IS-IS with SRv6 underlay tunnels created. The BGP internet prefixes and L3VPN prefixes with uSIDs can resolve over the underlay tunnels. These prefixes can be intent (with a color-community attached to them). The prefixes are resolved over SRv6 underlay tunnels installed in *junos-rti-tc-<color>.inet6.3* table, where color corresponds to the color in the color-community advertised with the prefix. If the prefix resolution does not happen over the *junos-rti-tc-<color>.inet6.3* table, it can resolve over flex-algorithm underlay tunnel installed in inet6.3 table, thus providing a fallback mechanism.
Support for programming uN SID and forwarding packets based on the uN SID routes	We support programming uN SID routes in inet6.0 table. This facilitates forwarding of packets that have uN SIDs as a part of their destination address. Note that packets having Segment Routing Header (SRH) is not supported currently.
Supported SID Functionalities	uN SID shift and lookup (flavor type `none`) uDT for IPv4 uDT for IPv6 uDT for IPv4 and IPv6
Locator summarization and leaking	Locator summarization and leaking is an advantage of SRv6 over SR-MPLS. An L1-L2 may summarize locators from other levels and leak them to another level. Locator leaking can happen even without summarization.

Configuring SRv6 in JCNR

Consider the following topology:

Figure 3: Topology Diagram



We want to enable communication between hosts 10.1.1.1/32 and 10.2.2.2/32 via an SRv6 tunnel. PE1 is an ingress Cloud-Native Router node, P is transit Cloud-Native Router node while PE2 is an egress Cloud-Native Router node. PE1, P, and PE2 advertise their uSIDs (shown in the figure) via IS-IS along with overlay prefixes via BGP to each other. CE1 initiates the packet flow to the ingress node (PE1). PE1 encapsulates the source packet with the SRv6 header while setting the destination address to the uSID of the egress node (PE2). Since Cloud-Native Router is the ingress node in this topology, the transit node (P) simply forwards the IPv6 packet. The egress node (PE2) is configured with a micro-service-SID (uDT) to decapsulate the packet and lookup a specific IP table to route the packet to CE2.

The configuration of SRv6 in Cloud-Native Router includes the micro-SID block configuration, micro-SID locator configuration, micro-node-SID (uN) configuration in IS-IS and micro-service-SID (uDT) configuration in BGP.

"Configure the JCNR control plane" on page 329 for SRv6. For brevity, we will cover the configuration for the egress Cloud-Native Router node (PE2) in this example. The configuration for other nodes is similar.

1. Configure the micro-SID block and optionally the maximum number of local static SIDs (default value is 0, must be configured if using static uSIDs):

```
user@PE2# set routing-options source-packet-routing srv6 block blk16_1 2001:db8::/32
user@PE2# set routing-options source-packet-routing srv6 block blk16_1 local-micro-sid
maximum-static-sids 2000
```

2. Configure the micro-SID locator:

```
user@PE2# set routing-options source-packet-routing srv6 locator myloc 2001:db8:4600::/48
user@PE2# set routing-options source-packet-routing srv6 locator myloc micro-sid block-name
blk16_1
user@PE2# set routing-options source-packet-routing srv6 locator myloc micro-sid flavor none
```

3. Configure micro-node-SID (uN SID) in IS-IS to advertise locator TLV and micro-node-SID:

```
user@PE2# set protocols isis source-packet-routing srv6 locator myloc micro-node-sid
```

4. There are two ways to configure micro-service-SIDs. The first is only BGP, which enables one set of dt4, dt6, and dt46 uSIDs to be configured per BGP instance. The uSID can either be static or dynamically allocated. The second way is to configure an export policy to specify the micro-service-SID used by a prefix or the locator to derive the micro-service-SID from. BGP is configured via the `non-default` keyword. An example for each method is provided below. You must use only one of them for a set of dt4, dt6, and dt46 uSIDs.

   a. Static default micro-service-SID for IPv4 services. Note that the static uSID must be in the `maximum-static-sids` range defined in Step 1:

   ```
   user@PE2# set routing-instances pe-2 protocols bgp source-packet-routing srv6 locator
   myloc micro-dt4-sid 0xF831
   ```

   b. Set auto-allocated default micro-service-SID for IPv4 services:

   ```
   user@PE2# set routing-instances pe-2 bgp source-packet-routing srv6 locator myloc micro-
   dt4-sid
   ```

   c. Set non-default micro-service-SID with BGP export policy:

   ```
   user@PE2# set routing-instances pe-2 bgp source-packet-routing srv6 locator myloc micro-
   dt4-sid non-default
   ```

   ```
   user@PE2# set policy-options policy-statement EXPORT_BGP_SRV6 term 1 then srv6 locator
   myloc
   user@PE2# set policy-options policy-statement EXPORT_BGP_SRV6 term 1 then srv6 micro-dt4-
   sid
   ```

5. Configure BGP to advertise SRv6 service. An example for family `inet` is provided below:

```
user@PE2# set protocols bgp group CNIv6 family inet unicast advertise-srv6-service
user@PE2# set protocols bgp group CNIv6 family inet unicast accept-srv6-service
```

Verify SRv6 Configuration on JCNR

The following commands can be used to verify the SRv6 configuration on "cRPD" on page 329:

```
user@host> show srv6 block blk16_1
Block: blk16_1
  Block Prefix: 2001:db8::, Block length: 32, Micro-sid length: 16
  Global Micro SIDs:
    Static SID range: 0x0-0xDFFF, Dynamic SID range: -
    Allocated static SID count: 1, Allocated dynamic SID count: 0
    Available static SID count: 57343, Available dynamic SID count: 0
  Local Micro SIDs:
    Static SID range: 0xF830-0xFFFF, Dynamic SID range: 0xE000-0xFFFF
    Allocated static SID count: 0, Allocated dynamic SID count: 0
    Available static SID count: 2000, Available dynamic SID count: 8192
```

```
user@host> show srv6 locator
Locator: myloc
  Locator prefix: 2001:db8:4600::, Locator length: 48
  Block length: 32, Node length: 16
  Function length: 16, Argument length: 0
  Micro SID Locator, Flavor [ None ]
  Micro SID Block Name: blk16_1
```

```
user@host> show isis overview
Instance: master
  Router ID: 10.1.1.1
  IPv6 Router ID: ::10.1.1.1
  ...
  Source Packet Routing (SPRING): Enabled
    Node Segments: Disabled
    SRv6: Enabled
      Locator: 2001:db8:4600::/48, Algorithm: 0
        micro-node-SID: 2001:db8:4600::, Flavor: None
...
```

```
user@host> show isis database extensive
IS-IS level 1 link-state database:
```

```
...
SRv6 Locator: 2001:db8:4600::/48, Metric: 0, MTID: 0, Flags: 0x0, Algorithm: 0
      SRv6 SID: 2001:db8:4600::, Flavor: None
      sid-structure-sub-sub-tlv: Block-length:32, Node-length:16
...
```

Verify Packet Flow via Cloud-Native Router Forwarding Plane

Verify the traffic flow via the "vRouter" on page 331 on each PE node:

1. Ingress SRv6 node (PE1)

```
[user@PE1 /]# flow --match 10.1.1.1
Flow table(size 161218560, entries 629760)

Entries: Created 300 Added 300 Deleted 400 Changed 600Processed 300 Used Overflow entries 0
(Created Flows/CPU: 0 0 0 0 0 0 0 0 0 0 0 72 87 72 69)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead
 Stats:Packets/Bytes

Listing flows matching ([10.1.1.1]:*)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
   231600<=>349580       10.1.1.1:1024                                      6 (98)
                         10.2.2.2:1024
(Gen: 3, K(nh):98, Action:F, Flags:, TCP:, QOS:-1, S(nh):0,  Stats:77632/8228992,
 SPort 63335, TTL 0, Sinfo 22.0.0.0)

   349580<=>231600       10.2.2.2:1024                                     6 (98)
                         10.1.1.1:1024
(Gen: 3, K(nh):98, Action:F, Flags:, TCP:, QOS:-1, S(nh):623,  Stats:0/0,
 SPort 63397, TTL 0, Sinfo 0.0.0.0)
```

```
[user@PE1 /]# rt --get 10.2.2.2/32 --vrf 98
Match 10.2.2.2/32 in vRouter inet4 table 0/98/unicast
```

```
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/98/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
10.2.2.0/24           0          LPT          -            623        -
```

```
[user@PE1 /]# nh --get 623
Id:623        Type:Tunnel         Fmly:AF_INET6  Rid:0  Ref_cnt:2         Vrf:0
              Flags:Valid, Policy, Etree Root, SRv6,
              Oif:2 Len:14 Data:40 a6 b7 a0 ef f1 50 7c 6f 48 9c 89 86 dd
              Sip: abcd:44:44:44::44
              Block Len:32 Block: 2001:db8::
              Number of Containers:1
              Container Dips:[1]: 2001:db8:4600:e001::
```

Note that the next hop for traffic to 10.2.2.2/32 is an SRv6 tunnel with Destination IP:`2001:db8:4600:e001::` (The uSID of egress node)

2. Transit SRv6 node (P)

```
[user@P /]# flow --match 2001:db8:4600:e001::
Flow table(size 161218560, entries 629760)

Entries: Created 300 Added 300 Deleted 400 Changed 600Processed 300 Used Overflow entries 0
(Created Flows/CPU: 0 0 0 0 0 0 0 0 0 0 0 72 87 72 69)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead
 Stats:Packets/Bytes

Listing flows matching ([2001:db8:4600:e001::]:*)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
   137640<=>238208       abcd:44:44:44::44:0                                 4 (0)
                         2001:db8:4600:e001:::0
(Gen: 1, K(nh):0, Action:F, Flags:, QOS:-1, S(nh):0,  Stats:81560/11907760,
 SPort 53031, TTL 0, Sinfo 0.0.0.0)
```

```
  238208<=>137640      2001:db8:4600:e001:::0                             4 (0)
                       abcd:44:44:44::44:0
(Gen: 1, K(nh):0, Action:F, Flags:, QOS:-1, S(nh):0,  Stats:0/0,  SPort 55361,
 TTL 0, Sinfo 0.0.0.0)
```

```
[user@P /]# rt --get 2001:db8:4600:e001::/128 --vrf 0 --family inet6
rt --get fcbb:bb01:4600:e001::/128 --vrf 0 --family inet6
Match fcbb:bb01:4600:e001::/128 in vRouter inet6 table 0/0/unicast

Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet6 routing table 0/0/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
2001:db8:4600::/48     0            T           -              80        -
```

```
[user@P /]# nh --get 80
Id:80          Type:Encap          Fmly:AF_INET/6  Rid:0  Ref_cnt:11         Vrf:0
               Flags:Valid, Policy, Etree Root,
               EncapFmly:0806 Oif:7 Len:14
               Encap Data: 50 7c 6f 48 83 79 40 a6 b7 a0 f9 3b
```

Note that P only forwards the IPv6 packet.

3. Egress SRv6 node (PE2)

```
[root@PE2 /]# rt --get 2001:db8:4600:e001::/128 --vrf 0 --family inet6
Match 2001:db8:4600:e001::/128 in vRouter inet6 table 0/0/unicast

Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet6 routing table 0/0/unicast
Destination               PPL        Flags        Label         Nexthop    Stitched MAC(Index)
2001:db8:4600:e001::/80   0            T           -             72         -
```

```
[root@PE2 /]# nh --get 72
Id:72          Type:Vrf_Translate  Fmly:AF_INET6  Rid:0  Ref_cnt:7          Vrf:19
```

```
              Flags:Valid, Etree Root, SRv6,
              Vrf:19
```

```
[root@PE2 /]# rt --get 10.2.2.2/32 --vrf 19
Match 10.2.2.2/32 in vRouter inet4 table 0/19/unicast

Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/19/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
10.2.2.2/32           0           PT          -            776         -
```

```
[root@PE2 /]# nh --get 776
Id:776        Type:Encap          Fmly:AF_INET/6  Rid:0  Ref_cnt:2          Vrf:19
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:21 Len:14
              Encap Data: 00 10 94 00 04 1e 50 7c 6f 48 77 64
```

Note that the Egress PE looks up for the uSID IPv6 address in the default table. The exact match next-hop is configured to `Vrf_Translate`. It looks up `10.2.2.2/32` destination IP address in the specified VRF and routes the packet to CE2.

Topology-Independent Loop-Free Alternates (TI-LFA)

SUMMARY

The Juniper cloud-native router (JCNR) supports Topology-Independent Loop-Free Alternates (TI-LFA) with fast reroute (FRR) for SR-MPLS implementations. It supports protection against link-failure that is detected by the Poll Mode Drivers (PMD).

IN THIS SECTION

TI-LFA Overview

Cloud-Native Router supports implementing Topology-Independent Loop-Free Alternates (TI-LFA) with fast reroute (FRR) in SR-MPLS. In the sections below we will look at Cloud-Native Router TI-LFA implementation in detail.

Fast Reroute (FRR)

Fast Reroute (FRR) technology is used in modern routing protocols to minimize disruption and packet loss in the event of a network failure. Unlike traditional routing protocols, FRR speeds up the convergence process by proactively pre-computing backup paths for critical traffic flows. The backup paths are readily available and instantly activated upon failure, ensuring minimum loss of data communication.

Loop-Free Alternates (LFAs) play a critical role in FRR implementations. LFAs are pre-calculated, loop-free backup paths that are leveraged by FRR protocols to quicky switch traffic to the chosen backup path.

There are two key aspects to FRR:

- Detection:
  - Fault Detection—Rapidly identify link or node failures within a network through continuous monitoring of network elements.
  - Detection Methods—Applying one of the various methods for fault detection, including proactive methods such as periodic link probing or reactive methods that sense failure when it occurs. In some cases Bidirectional Forwarding Detection (BFD) may be used for quick fault detection. Cloud-Native Router detects failure by monitoring the link status events from the Poll Mode Driver (PMD).
- Handling:
  - Precomputed Backup Paths—Using predetermined and optimised backup paths that are quickly activated when a failure is detected.
  - Fast Switchover—Network devices participating in the FRR process quickly switching to precomputed backup paths to minimize the impact on the network's forwarding capabilities.

The control plane is responsible for preparing the backup paths and detecting when failure occurs. It also re-calculates the best path based on the updated network topology and propagates information about the new path to other routers in the network. This ensures the network converges to the optimal path in the long run. The data plane is responsible for quickly failing over to the backup path in the event of a failure detection. Typically, with LFA implementations, pre-configured forwarding entries or next-hop information for the LFA are already installed on the router. The data plane also adjusts its forwarding tables when it receives a new path from the control plane upon re-calculation. Efficient communication between the control and data planes is critical to minimize lag in traffic reroute, ensuring reduced downtime, improved performance and enhanced reliability.

Topology-Independent Loop-Free Alternates (TI-LFA)

IS-IS and OSPF can calculate LFA backup paths in a plain IP network. However, the LFA feature requires any backup path to be guaranteed loop-free. For this reason, in a plain IP network, LFA cannot offer backup paths to every single known destination. LFA only offers partial topology coverage. TI-LFA is a topology independent implementation of LFA. TI-LFA can push a Segment ID stack that can navigate around any potential loops along the backup path. In other words, backup paths can be calculated independent of the topology. It defines LFAs based on the traffic flow itself and can function effectively regardless of the underlying network layout. TI-LFA uses the pre-computed post-convergence path of the routing protocol. TI-LFA finds the path that would be calculated in the event of a particular link or node failure, and uses that exact path as the backup path. This is not always possible in regular LFA, due to the requirement for a loop-free backup path. By using the post-convergence path, TI-LFA reduces jitter during failover and the network operator only needs to ensure the network has enough capacity to carry the traffic on the post-convergence path after a failure. TI-LFA has multiple advantages:

- Simplified configuration—TI-LFA automatically computes backup paths, eliminating the need to manually configurate LFAs for each network element.
- Faster failover—TI-LFA uses the pre-computed post-convergence path of the routing protocol that enables it to activate backup paths significantly faster than traditional LFAs.
- Improved scalability—TI-LFA scales efficiently in large and complex networks because it is topology independent in contrast to LDP and RSVP which require additional state to create backup paths.

Cloud-Native Router supports TI-LFA for SR-MPLS implementations with link failure protection . You can read more about TI-LFA in the Junos documentation.

TI-LFA Implementation in JCNR

An IGP identifies the primary and post-convergence (backup for TI-LFA) paths for a prefix based on its criterion. The paths are associated with a weight metric to signify priority (numerically lower the weight, higher the priority). The Cloud-Native Router control plane (cRPD) sends the primary and backup path to the data plane via the vRouter agent. The vRouter data plane implements FRR by identifying the primary path and quickly switching over to backup path if a link failure was detected.

Key points to note about the Cloud-Native Router TI-LFA implementation:

- TI-LFA is supported for SR-MPLS on IS-IS implementations.
- TI-LFA is supported for SR-MPLS on OSPF implementations (*Juniper Technology Preview* Feature).

- TI-LFA is supported when Cloud-Native Router is deployed as head-end, transit or egress node in an SR-MPLS domain.
- Only TI-LFA protection against link failure is supported.
- FRR is triggered based on link status events detected by the Poll Mode Drivers (PMD).
- One primary and one backup path is supported.
- Both physical and bond interfaces are supported (FRR is triggered for bond interfaces only if all the links in the bond are down).

TI-LFA Configuration (SR-MPLS on IS-IS)

IN THIS SECTION

Steps to configure TI-LFA

The following example requires that you navigate various levels in the configuration hierarchy. For information about navigating the CLI, see Using the CLI Editor in Configuration Mode in the Junos OS CLI User Guide.

1. Configure IS-IS.

```
set protocols isis interface enp3s0 level 2
set protocols isis interface enp3s0 hello-padding disable
set protocols isis interface enp3s0 point-to-point
set protocols isis interface enp5s0 level 2
set protocols isis interface enp5s0 hello-padding disable
set protocols isis interface enp5s0 point-to-point
set protocols isis interface enp7s0 level 2
set protocols isis interface enp7s0 hello-padding disable
set protocols isis interface enp7s0 point-to-point
set protocols isis interface lo0.0
```

2. Configure Segment Routing for IS-IS. In this example we have used node SID and segment routing global block (SRGB) range.

```
set protocols isis source-packet-routing node-segment ipv4-index 2
set protocols isis source-packet-routing srgb start-label 10000
set protocols isis source-packet-routing srgb index-range 25000
set protocols isis source-packet-routing explicit-null
set protocols isis backup-spf-options use-source-packet-routing
```

3. Enable TI-LFA for IS-IS. Link protection is supported by default. Ensure `maximum-backup-paths` is set to 1 since Cloud-Native Router supports only one backup path currently.

```
set protocols isis backup-spf-options use-post-convergence-lfa maximum-backup-paths 1
```

4. Configure to install backup route along the link-protecting post-convergence path on the interfaces.

```
set protocols isis interface enp3s0 level 2 post-convergence-lfa
set protocols isis interface enp5s0 level 2 post-convergence-lfa
set protocols isis interface enp7s0 level 2 post-convergence-lfa
```

Verify Configuration

- To verify the TI-LFA configuration you can "access the cRPD shell" on page 329 and verify you have the primary and backup path for a destination. For example for the destination `192.168.7.2`, the show route output below shows the primary path via `192.168.2.2` and backup path via `192.168.3.2` with the SR-MPLS labels :

```
user@host> show route 192.168.7.2

ce1.inet.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

192.168.7.2/32     *[BGP/170] 00:03:20, localpref 100, from 2.2.2.2
                      AS path: I, validation-state: unverified
                    >  to 192.168.2.2 via ens1f1, Push 66
                       to 192.168.3.2 via ens1f2, Push 66, Push 16002, Push 16003(top)
```

- You can also verify the routes for the destination in the vRouter forwarding table by "accessing the vRouter pod shell" on page 331. Notice the output below shows the route to destination

`192.168.7.2` is a composite next hop (NH). The composite next-hop has sub next-hop `79` and `80` with next-hop `79` having a lower ECMP weight of 1, signifying it as a primary path:

```
# rt --get 192.168.7.2/32 --vrf 11 --family inet
Match 192.168.7.2/32 in vRouter inet4 table 0/11/unicast

Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/11/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
192.168.7.2/32          0          PT           -              91        -
[root@node-warthog-29 /]# nh --get 91
Id:91          Type:Composite      Fmly: AF_INET  Rid:0  Ref_cnt:2          Vrf:11
              Flags:Valid, Policy, Weighted Ecmp, FRR, Etree Root,
              Sub NH(label): 79(66) 80(66)
              ECMP Weights: 1, 61440,
              FRR State: 0 -> 1 FRR Updates: 0
              FRR State Valid List: 1, 1,

Id:79          Type:Tunnel         Fmly: AF_MPLS  Rid:0  Ref_cnt:23         Vrf:0
              Flags:Valid, Policy, Etree Root, MPLS,
              Oif:3 Len:14 Data:40 a6 b7 c4 2a fa 40 a6 b7 6f 3a c5 88 47 Number of
Transport Labels:0

Id:80          Type:Tunnel         Fmly: AF_MPLS  Rid:0  Ref_cnt:23         Vrf:0
              Flags:Valid, Policy, Etree Root, MPLS,
              Oif:4 Len:14 Data:50 7c 6f 48 83 f4 40 a6 b7 6f 3a c6 88 47 Number of
Transport Labels:2 Transport Labels:16002, 16003,
```

TI-LFA Configuration (SR-MPLS on OSPF)

IN THIS SECTION

NOTE: This is a "Juniper Technology Previews (Tech Previews)" on page 333 Feature.

Steps to configure TI-LFA

The following example requires that you navigate various levels in the configuration hierarchy. For information about navigating the CLI, see Using the CLI Editor in Configuration Mode in the Junos OS CLI User Guide.

1. Configure OSPF.

```
set protocols ospf area 0.0.0.0 interface enp3s0 interface-type p2p
set protocols ospf area 0.0.0.0 interface enp3s0 hello-interval 30
set protocols ospf area 0.0.0.0 interface enp3s0 dead-interval 90
set protocols ospf area 0.0.0.0 interface enp3s0 post-convergence-lfa
set protocols ospf area 0.0.0.0 interface enp5s0 interface-type p2p
set protocols ospf area 0.0.0.0 interface enp5s0 hello-interval 30
set protocols ospf area 0.0.0.0 interface enp5s0 dead-interval 90
set protocols ospf area 0.0.0.0 interface enp5s0 post-convergence-lfa
set protocols ospf area 0.0.0.0 interface enp7s0 interface-type p2p
set protocols ospf area 0.0.0.0 interface enp7s0 hello-interval 30
set protocols ospf area 0.0.0.0 interface enp7s0 dead-interval 90
set protocols ospf area 0.0.0.0 interface enp7s0 post-convergence-lfa
set protocols ospf area 0.0.0.0 interface lo0.0
```

2. Configure Segment Routing for OSPF. In this example we have used adjacency SID and segment routing global block (SRGB) range.

```
set protocols ospf source-packet-routing adjacency-segment hold-time 180000
set protocols ospf source-packet-routing prefix-segment prefix-sid
set protocols ospf source-packet-routing explicit-null
set protocols ospf source-packet-routing srgb start-label 10000
set protocols ospf source-packet-routing srgb index-range 25000
set protocols ospf backup-spf-options use-source-packet-routing
```

3. Enable TI-LFA for OSPF. Link protection is supported by default. Ensure `maximum-backup-paths` is set to 1.

```
set protocols ospf backup-spf-options use-post-convergence-lfa maximum-backup-paths 1
```

Access Control Lists (Firewall Filters)

SUMMARY

Read this topic to learn about the Layer 3-Layer 4 access control lists (firewall filters) in the cloud-native router.

IN THIS SECTION

Juniper Cloud-Native Router Release supports stateless firewall filters. Firewall filters provide a means of protecting the cloud-native router from excessive traffic transiting the router to a network destination or destined for the Routing Engine. A stateless firewall filter, also known as an access control list (ACL), does not statefully inspect traffic. Instead, it evaluates packet contents statically and does not keep track of the state of network connections. The basic purpose of a stateless firewall filter is to enhance security through the use of packet filtering. Packet filtering enables you to inspect the components of incoming or outgoing packets and then perform the actions you specify on packets that match the criteria you specify. The typical use of a stateless firewall filter is to protect the Routing Engine processes and resources from malicious or untrusted packets.

To influence which packets are allowed to transit the system and to apply special actions to packets as necessary, you can configure a sequence of one or more packet-filtering rules, called *filter terms*. A filter term specifies *match conditions* to use to determine a match and *actions* to take on a matched packet. A stateless firewall filter enables you to manipulate any packet of a particular protocol family, including fragmented packets, based on evaluation of Layer 3 and Layer 4 header fields. Please review the Stateless Firewall Filter Overview topic for more information.

Cloud-Native Router also supports "Layer-2 access control lists (firewall filter for bridge family)" on page 45.

> **NOTE**: In Cloud-Native Router you can apply a stateless firewall filter to an ingress interface only. The supported interfaces types include a fabric interface, sub-interface, pod interface and an `irb` interface.

> **NOTE**: Cloud-Native Router supports a maximum number of 16 filters per family and 16 terms per filter.

Supported Protocol Families

Table 17: Firewall Filter Protocol Families

Type of Traffic to be Filtered	Configuration Statement
Internet Protocol version 4 (IPv4)	family inet
Internet Protocol version 6 (IPv6)	family inet6
MPLS	family mpls

Supported Match Conditions and Actions (IPv4 and IPv6)

Cloud-Native Router supports the IPv4 and IPv6 standard firewall filter with the match conditions and actions provided in the table.

Table 18: Firewall Filter Match Conditions for IPv4 Traffic

Match Condition	Description
destination-address *address*	Match the IPv4 destination address field. You can provide a prefix with an optional subnet mask.

Table 18: Firewall Filter Match Conditions for IPv4 Traffic *(Continued)*

Match Condition	Description
destination-port *number*	Match the UDP or TCP destination port field. When configuring port based matches you must also configure the `protocol udp` or `protocol tcp` match statement in the same filter term. Matching only on the port value can result in unexpected matches. In place of the numeric value, you can specify one of the following text synonyms (the port numbers are also listed): `afs` (1483), `bgp` (179), `biff` (512), `bootpc` (68), `bootps` (67), `cmd` (514), `cvspserver` (2401), `dhcp` (67), `domain` (53), `eklogin` (2105), `ekshell` (2106), `exec` (512), `finger` (79), `ftp` (21), `ftp-data` (20), `http` (80), `https` (443), `ident` (113), `imap` (143), `kerberos-sec` (88), `klogin` (543), `kpasswd` (761), `krb-prop` (754), `krbupdate` (760), `kshell` (544), `ldap` (389), `ldp` (646), `login` (513), `mobileip-agent` (434), `mobilip-mn` (435), `msdp` (639), `netbios-dgm` (138), `netbios-ns` (137), `netbios-ssn` (139), `nfsd` (2049), `nntp` (119), `ntalk` (518), `ntp` (123), `pop3` (110), `pptp` (1723), `printer` (515), `radacct` (1813), `radius` (1812), `rip` (520), `rkinit` (2108), `smtp` (25), `snmp` (161), `snmptrap` (162), `snpp` (444), `socks` (1080), `ssh` (22), `sunrpc` (111), `syslog` (514), `tacacs` (49), `tacacs-ds` (65), `talk` (517), `telnet` (23), `tftp` (69), `timed` (525), `who` (513), or `xdmcp` (177).
source-address *address*	Match the IPv4 address of the source node sending the packet. You can provide a prefix with an optional subnet mask.
source-port *number*	Match the UDP or TCP source port field. When configuring port based matches you must also configure the `protocol udp` or `protocol tcp` match statement in the same filter term. Matching only on the port value can result in unexpected matches. In place of the numeric value, you can specify one of the text synonyms listed with the `destination-port number` match condition.
protocol *number*	Match the IP protocol type field. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `ah` (51), `dstopts` (60), `egp` (8), `esp` (50), `fragment` (44), `gre` (47), `hop-by-hop` (0), `icmp` (1), `icmp6` (58), `icmpv6` (58), `igmp` (2), `ipip` (4), `ipv6` (41), `ospf` (89), `pim` (103), `rsvp` (46), `sctp` (132), `tcp` (6), `udp` (17), or `vrrp` (112).

Table 18: Firewall Filter Match Conditions for IPv4 Traffic *(Continued)*

Match Condition	Description
tcp-flags *value*	Match one or more of the low-order 6 bits in the 8-bit TCP flags field in the TCP header. To specify individual bit fields, you can specify the following text synonyms or hexadecimal values: • `fin` (0x01) • `syn` (0x02) • `rst` (0x04) • `push` (0x08) • `ack` (0x10) • `urgent` (0x20) In a TCP session, the SYN flag is set only in the initial packet sent, while the ACK flag is set in all packets sent after the initial packet. You can string together multiple flags using the bit-field logical operators. If you configure this match condition, we recommend that you also configure the `protocol tcp` match statement in the same term to specify that the TCP protocol is being used on the port.
icmp-type *number*	Match the ICMP message type field. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): `echo-reply` (0), `echo-request` (8), `info-reply` (16), `info-request` (15), `mask-request` (17), `mask-reply` (18), `parameter-problem` (12), `redirect` (5), `router-advertisement` (9), `router-solicit` (10), `source-quench` (4), `time-exceeded` (11), `timestamp` (13), `timestamp-reply` (14), or `unreachable` (3).

Table 19: Firewall Filter Match Conditions for IPv6 Traffic

Match Condition	Description
destination-address *address*	Match the IPv6 destination address field. You can provide a prefix with an optional subnet mask.
destination-port *number*	Match the UDP or TCP destination port field. When configuring port based matches you must also configure the `protocol udp` or `protocol tcp` match statement in the same filter term. Matching only on the port value can result in unexpected matches. In place of the numeric value, you can specify one of the following text synonyms (the port numbers are also listed): `afs` (1483), `bgp` (179), `biff` (512), `bootpc` (68), `bootps` (67), `cmd` (514), `cvspserver` (2401), `dhcp` (67), `domain` (53), `eklogin` (2105), `ekshell` (2106), `exec` (512), `finger` (79), `ftp` (21), `ftp-data` (20), `http` (80), `https` (443), `ident` (113), `imap` (143), `kerberos-sec` (88), `klogin` (543), `kpasswd` (761), `krb-prop` (754), `krbupdate` (760), `kshell` (544), `ldap` (389), `ldp` (646), `login` (513), `mobileip-agent` (434), `mobilip-mn` (435), `msdp` (639), `netbios-dgm` (138), `netbios-ns` (137), `netbios-ssn` (139), `nfsd` (2049), `nntp` (119), `ntalk` (518), `ntp` (123), `pop3` (110), `pptp` (1723), `printer` (515), `radacct` (1813), `radius` (1812), `rip` (520), `rkinit` (2108), `smtp` (25), `snmp` (161), `snmptrap` (162), `snpp` (444), `socks` (1080), `ssh` (22), `sunrpc` (111), `syslog` (514), `tacacs` (49), `tacacs-ds` (65), `talk` (517), `telnet` (23), `tftp` (69), `timed` (525), `who` (513), or `xdmcp` (177).
source-address *address*	Match the IPv6 address of the source node sending the packet. You can provide a prefix with an optional subnet mask.
source-port *number*	Match the UDP or TCP source port field. When configuring port based matches you must also configure the `protocol udp` or `protocol tcp` match statement in the same filter term. Matching only on the port value can result in unexpected matches. In place of the numeric value, you can specify one of the text synonyms listed with the `destination-port number` match condition.

Table 19: Firewall Filter Match Conditions for IPv6 Traffic *(Continued)*

Match Condition	Description
tcp-flags *value*	Match one or more of the low-order 6 bits in the 8-bit TCP flags field in the TCP header. To specify individual bit fields, you can specify the following text synonyms or hexadecimal values: • fin (0x01) • syn (0x02) • rst (0x04) • push (0x08) • ack (0x10) • urgent (0x20) In a TCP session, the SYN flag is set only in the initial packet sent, while the ACK flag is set in all packets sent after the initial packet. You can string together multiple flags using the bit-field logical operators.
icmp-type *message-type*	Match the ICMP message type field. In place of the numeric value, you can specify one of the following text synonyms (the field values are also listed): certificate-path-advertisement (149), certificate-path-solicitation (148), destination-unreachable (1), echo-reply (129), echo-request (128), home-agent-address-discovery-reply (145), home-agent-address-discovery-request (144), inverse-neighbor-discovery-advertisement (142), inverse-neighbor-discovery-solicitation (141), membership-query (130), membership-report (131), membership-termination (132), mobile-prefix-advertisement-reply (147), mobile-prefix-solicitation (146), neighbor-advertisement (136), neighbor-solicit (135), node-information-reply (140), node-information-request (139), packet-too-big (2), parameter-problem (4), private-experimentation-100 (100), private-experimentation-101 (101), private-experimentation-200 (200), private-experimentation-201 (201), redirect (137), router-advertisement (134), router-renumbering (138), router-solicit (133), or time-exceeded (3).

Table 20: Firewall Filter Actions (IPv4 and IPv6)

Type of Action	Description	Supported actions
Terminating	Halts all evaluation of a firewall filter for a specific packet. The router (or switch) performs the specified action, and no additional terms are used to examine the packet. You can specify only one *terminating action* in a firewall filter term. If you try to specify more than one *terminating action* within the filter term then the latest *terminating action* will replace the existing *terminating action*. You can, however, specify one terminating action with one or more *nonterminating actions* in a single term. For example, within a term, you can specify `accept` with `count`. Regardless of the number of terms that contain terminating actions, once the system processes a terminating action within a term, processing of the entire firewall filter halts.	• `accept` —Accept the packet • `discard` —Discard a packet silently, without sending an Internet Control Message Protocol (ICMP) message. Discarded packets are available for logging and sampling.

Table 20: Firewall Filter Actions (IPv4 and IPv6) ***(Continued)***

Type of Action	Description	Supported actions

Nonterminating	Performs other functions on a packet (such as incrementing a counter, logging information about the packet header, sampling the packet data, or sending information to a remote host using the system log functionality), but any additional terms are used to examine the packet.
	
	Note: Cloud-Native Router supports `count` as a nonterminating action only when added along with a terminating action.
		• count *counter-name*
		
		• log—Log the packet header information in a buffer within the Packet Forwarding Engine. You can access this information by issuing the `show firewall log` command at the command-line interface (CLI).
		
		• syslog—Log the packet to the system log file.
		
		NOTE: Cloud-Native Router is preconfigured with the following syslog configuration:
		
		`set system syslog file jcnr-firewall.log any any`
		`set system syslog file jcnr-firewall.log match-strings "JCNR-FIREWALL"`
		
		You must additionally configure syslog as follows:
		
		`set system syslog file messages_firewall_any match-strings "JCNR-FIREWALL"`
		
		• routing-instance routing-instance-name—Direct packets to the specified routing instance.
		
		• forwarding-class *class-name*—Classify the packet to the named forwarding class.

Table 20: Firewall Filter Actions (IPv4 and IPv6) *(Continued)*

Type of Action	Description	Supported actions
		• policer *policer-name*—Name of policer to use to rate-limit traffic.

Supported Match Conditions and Actions (MPLS)

Cloud-Native Router supports the MPLS standard firewall filter with the match conditions and actions provided in the table.

Table 21: Firewall Filter Match Conditions for MPLS Traffic

Match Condition	Description
exp *number*	Experimental (EXP) bit number or range of bit numbers in the MPLS header of a packet. For number, you can specify one or more values from 0 through 7 in binary, decimal or hexadecimal format, as given below: • A single EXP bit—for example, **exp 3** • Several EXP bits—for example, **exp 0,4** • A range of EXP bits—for example, **exp [0-5]**.
label *number*	MPLS label value or range of label values in the MPLS header of a packet. For number, you can specify one or more values from 0 through 1048575 in decimal or hexadecimal format, as given below: • A single label—for example, **label 3** • Several labels—for example, **label 0,4** • A range of labels—for example, **label [0-5]**

Table 22: Firewall Filter Actions (MPLS)

Type of Action	Description	Supported actions
Terminating	Halts all evaluation of a firewall filter for a specific packet. The router (or switch) performs the specified action, and no additional terms are used to examine the packet. You can specify only one *terminating action* in a firewall filter term. If you try to specify more than one *terminating action* within the filter term then the latest *terminating action* will replace the existing *terminating action*. You can, however, specify one terminating action with one or more *nonterminating actions* in a single term. For example, within a term, you can specify `accept` with `count` and `syslog`. Regardless of the number of terms that contain terminating actions, once the system processes a terminating action within a term, processing of the entire firewall filter halts.	• `accept` —Accept the packet. • `discard` —Discard a packet silently, without sending an Internet Control Message Protocol (ICMP) message. Discarded packets are available for logging and sampling.

Table 22: Firewall Filter Actions (MPLS) ***(Continued)***

Type of Action	Description	Supported actions

Nonterminating	Performs other functions on a packet (such as incrementing a counter, logging information about the packet header, sampling the packet data, or sending information to a remote host using the system log functionality), but any additional terms are used to examine the packet.
	
	Note: Cloud-Native Router supports `count`, `log`, `syslog`, and `routing-instance` as nonterminating actions only when added along with a terminating action.
		• count *counter-name*—Count the packet in the named counter.
		
		• log—Log the packet header information in a buffer within the Packet Forwarding Engine. You can access this information by issuing the `show firewall log` command at the command-line interface (CLI).
		
		• syslog—Log the packet to the system log file.
		
		NOTE: Cloud-Native Router is preconfigured with the following syslog configuration:
		
		`set system syslog file jcnr-firewall.log any any`
		`set system syslog file jcnr-firewall.log match-strings “JCNR-FIREWALL”`
		
		You must additionally configure syslog as follows:
		
		`set system syslog file messages_firewall_any match-strings “JCNR-FIREWALL”`

Configuration Example

NOTE: Use the *configlet resource* to configure the cRPD pods.

You can configure the Cloud-Native Router controller with a stateless firewall filter under the `firewall` hierarchy.

Configuration example for IPv4 family is provided below:

```
firewall {
    family inet {
        filter temp {
            term a {
                from {
                    source-address {
                        10.0.0.1/32;
                    }
                    destination-address {
                        10.0.0.2/32;
                    }
                    protocol icmp;
                    icmp-type echo-request;
                    source-port http;
                    destination-port bgp;
                    tcp-flags fin;
                }
                then {
                    count c1;
                    accept;
                }
            }
        }
    }
}
```

Configuration example for IPv6 family is provided below:

```
firewall {
    family inet6 {
        filter temp6 {
            term a {
                from {
                    source-address {
                        2001:db8::1/128;
                    }
                    destination-address {
```

```
                        2001:db8::1/128;
                    }
                    icmp-type echo-request;
                    source-port http;
                    destination-port bgp;
                    tcp-flags fin;
                }
                then {
                    count c1;
                    discard;
                }
            }
        }
    }
}
```

Configuration example for MPLS family is provided below:

```
firewall {
    family mpls {
        filter temp_mpls {
            term t1 {
                from {
                    exp 2-5;
                    label 1234-4321;
                }
                then {
                    accept;
                    count c1;
                    log;
                    syslog;
                }
            }
        }
    }
}
```

The filter will be applied to the ingress interface. The supported interfaces include a fabric interface, sub-interface, pod interface and an `irb` interface. The filter can be applied only on input for an interface:

```
user@host > show interfaces enp4s0
unit 0 {
    family inet {
        filter {
            input temp;
        }
        address 10.0.0.1/24;
    }
    family inet6 {
        filter {
            input temp6;
        }
    }
}
```

Troubleshooting

Cloud-Native Router Controller Commands

The following commands may be used on the Cloud-Native Router controller to view firewall information:

Display all firewall filters for family inet (IPv4)

```
user@host> show firewall family inet
Filter: temp
Counters:
Name
Bytes              Packets
c1
0                    0
c2
1532909                22500
Filter: temp 2
Counters:
Name
```

```
Bytes              Packets
c3
0                    0
c4
100                  100
```

Display a specific firewall filter for family inet

```
user@host> show firewall family inet filter temp
Filter: temp
Counters:
Name
Bytes              Packets
c1
0                    0
c2
1532909              22500
```

Display a specific counter for a firewall filter for family inet

```
user@host> show firewall family inet filter temp counter c2
Filter: temp
Counters:
Name
Bytes              Packets
c2
1532909              22500
```

Display all firewall filters for family inet6 (IPv6)

```
user@host> show firewall family inet6
Filter: temp6
Counters:
Name
Bytes              Packets
c1
0                    0
c2
1532909              22500
Filter: temp6_2
Counters:
```

```
Name
Bytes               Packets
c3
0                     0
c4
100                   100
```

Display all firewall filters for family mpls

```
user@host> show firewall family mpls
Filter: temp_mpls
Counters:
Name
Bytes               Packets
c1
0                     0
c2
1532909
22500
```

Display a specific firewall filter for family mpls

```
user@host> show firewall family mpls filter temp_mpls
Filter: temp_mpls
Counters:
Name
Bytes               Packets
c1
0                     0
c2
1532909                 22500
```

Clear the counter statistics:

```
clear firewall family name >> clear all counter statistics for inet, inet6 or mpls family
clear firewall family name filter name >> clear all counter statistics for a specific filter for
inet, inet6 or mpls family
clear firewall family name filter name count counter-name >> clear statistics for a specific
counter for a specific filter for inet, inet6, mpls family
```

View the firewall logs:

```
user@host> show firewall log
[JCNR-FIREWALL]: Mar 20 08:08:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:18:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:28:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:38:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
```

View syslog messages:

```
user@host> show log messages_firewall_any
[JCNR-FIREWALL]: Mar 20 08:08:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:18:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:28:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
[JCNR-FIREWALL]: Mar 20 08:38:45 hostname feb FW: ge-1/1/0.0 A icmp 192.168.207.222
192.168.207.223  0  0 (1 packets)
```

vRouter Commands

The following commands may be used on the vRouter to view the firewall configuration:

```
bash-5.1# acl --family  inet  --filter f4  --term  t4
=======================================
Filter: f4
=======================================
Term: t4
-----
Priority:     268420555
Dest IP:     10.0.0.1/32
Src IP:     10.0.0.2/32
Dst ports:     [179 - 179]
```

```
Src ports:     [179 - 179]
Action:     accept (n/a)
```

```
bash-5.1#  acl --list-filters --family mpls
=======================================
Filter: temp_mpls
=======================================
Term: t1
-----
Priority:     256
Label:        [2345 - 4092]
Exp:             [3 - 5]
Action:       discard (n/a)

=======================================
```

```
bash-5.1# acl --list-actions
[1] inet filter "temp_mpls": Counter "c1"
             Rx Packets: 2



[2] inet filter "temp_mpls": Counter "c2"
             Rx Packets: 1
```

Additional `acl` commands include the following:

```
acl --list-filters --family <inet/inet6/mpls> >>Lists the full acl table
acl --list-actions >>Shows the acl entry corresponding to filter name and term name
acl --family <inet/inet6/mpls> --filter <name> [--list-terms] >>Shows the ACL term list
acl --family <inet/inet6/mpls> --filter <name> [--term <name>] >>Shows the ACL term details
acl --family <inet/inet6/mpls> --filter <name> [--action <name>] >>Shows the ACL action details
acl --family <inet/inet6/mpls> --filter <name> [--action <name>] --clear >>Clears the ACL action
details
acl --help >>Prints the help messages
```

You can view the filter associated with an interface using the `vif --get` command:

```
bash-5.1# vif --get 5
Vrouter Interface Table
```

```
Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled


vif0/5      PCI: 0000:07:00.0 NH: 10 MTU: 9000
            Type:Physical HWaddr:02:8b:65:44:27:bd IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3L2Vof QOS:0 Ref:9
            RX device packets:8807  bytes:374638 errors:0
            RX port   packets:8806 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:07:00.0  Status: UP  Driver: 0000:07:00.0
            RX packets:8806  bytes:374596 errors:0
            TX packets:2  bytes:240 errors:0
            Drops:0
            TX queue  packets:2 errors:0
            TX port   packets:2 errors:0
            TX device packets:8  bytes:912 errors:0
            inet acl f1
            inet6 acl f1v6
```

```
bash-5.1# vif --get 1
Vrouter Interface Table


Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
```

```
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled

vif0/1      PCI: 0000:03:00.0 NH: 6 MTU: 9000
            Type:Physical HWaddr:02:d7:e7:8c:dc:0b IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3Vof Ref:9
            RX device packets:292  bytes:16204 errors:0
            RX port   packets:37 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:03:00.0  Status: UP  Driver: net_virtio
            RX packets:37  bytes:1582 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
            TX device packets:7  bytes:786 errors:0
            mpls acl testAclFabricWithMplsExp
```

IPsec Security Services

IN THIS SECTION

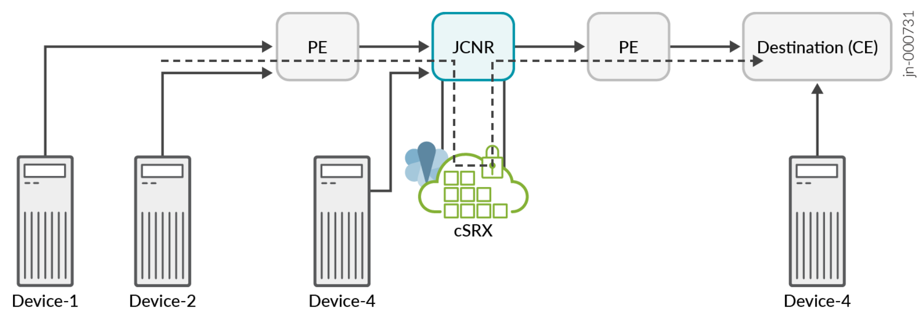
Read this topic to understand how the cloud-native router integrates with Juniper's cSRX to provide IPsec security services.

Juniper Cloud-Native Router (JCNR) offers containerized routing functionality for both cloud-based and on-premise 5G environments. There is a growing demand for integrating security services with JCNR.

This functionality can be achieved using host-based service chaining. The cloud-native router is integrated with Juniper's containerized SRX (cSRX) platform to provide security services such as IPsec.

Overview

Let us consider an IPsec security services use case with JCNR. In the figure below, the cloud-native router connects the provider edge (PE) routers in a service provider network. The customer edge (CE) routers or devices in the source network securely transfer data to the destination CEs via an IPsec tunnel. In the given scenario, the IPsec tunnel initiates from the cloud-native router's security services (cSRX) and terminates on the destination CEs. The cloud-native router and its peer PE provides the underlay connectivity to the IPsec tunnel.



The cloud-native router is chained with a security service instance (cSRX) in the same Kubernetes cluster. The cSRX instance runs as a pod service in L3 mode.

NOTE: A cloud-native router instance is service chained with only one instance of cSRX. It can support multiple IPSec tunnels.

Configuration

Cloud-Native Router can steer selective traffic through IPsec services. This is achieved by defining static routes to a destination via the cSRX interface. When the selective traffic is received on JCNR, it is forwarded to cSRX. The cSRX subjects the packet to IPsec encryption based on the configuration and forwards it to JCNR. Cloud-Native Router performs a route lookup in the untrust VRF and forwards traffic through the configured fabric interface. When IPsec packet is received from the remote end, Cloud-Native Router sends it to the security services. The cSRX decrypts the packet and forwards it to Cloud-Native Router on the trust interface. Cloud-Native Router then forwards the packet to the pod.

The IP addresses of the tunnel endpoints are configured in Cloud-Native Router as static routes and advertised through an IGP to the remote end.

You can customize the cSRX deployment by specifying a range of configuration parameters in the helm chart (values.yaml) for cSRX. Key configuration options include:

- `interfaceType`: This is the type of interface on the cSRX to connect to JCNR. Must be set to `vhost` only.
- `interfaceConfigs`: This is an array defining the interface IP address, gateway address and optionally routes. One of the interface IP must match the `localAddress` element in the `ipSecTunnelConfigs` array. This will be the interface in the untrust zone. The routes should contain prefixes to steer decrypted traffic to Cloud-Native Router and reachability route for IPSec gateway.
- `ipSecTunnelConfigs`: This is an array defining the IPsec configuration details such as ike-phase1, proposal, policy and gateway configuration. Traffic selector should contain traffic that is expected to be encrypted. You can define one or multiple tunnel configurations. You can either define the tunnel configuration using the installation helm chart or configure them post-deployment using a configlet.
- `jcnr_config`: This is an array defining the routes to be configured in Cloud-Native Router to steer traffic from Cloud-Native Router to cSRX and to steer IPsec traffic from the remote IPsec gateway to the cSRX to apply the security service chain.

Configuring cSRX via Helm Chart

The cSRX configurations are required to bring up its containers and pods at the time of installation. The IPSec tunnel configuration can be defined at the time of installation or later using a configlet. The `enableUserConfig` flag in the cSRX helm chart when set to true enables IPSec tunnel configuration via configlet.

By default the `enableUserConfig` flag is set to false in `values.yaml` and therefore requires `ipSecTunnelConfigs` to be configured at the time of installation. A sample interface and tunnel configuration for multiple tunnels is provided below:

```
enableUserConfig: false  # enable /disable user configuration

interfaceType: "vhost"

interfaceConfigs:
    - name: ge-0/0/0
      ip: 172.16.10.1/30          # should match ipSecTunnelConfigs localAddress if configured
      gateway: 172.16.10.2        # gateway configuration
      ip6: 2001:db8:172:16:10::1/64        # optional
      ip6Gateway: 2001:db8:172:16:10::2    # optional
      routes:                   # this field is optional
```

```
    - "172.17.10.0/24"
    - "172.18.10.0/24"
    instance_parameters:
      name: "untrust"
      type: "vrf"       # options include virtual-router or vrf
      vrfTarget: 10:10
  - name: ge-0/0/1
    ip: 192.168.1.1/30          # should match ipSecTunnelConfigs localAddress if configured
    gateway: 192.168.1.2        # gateway configuration
    ip6: 2001:db8:192:168:1::1/64                     # optional
    ip6Gateway: 2001:db8:192:168:1::2               # optional
    routes:                   # this field is optional
    - "10.111.1.0/24"
    - "10.112.1.0/24"
    instance_parameters:
      name: "trust"
      type: "vrf"       # options include virtual-router or vrf
      vrfTarget: 11:11

ipSecTunnelConfigs:        # untrust
  - interface: ge-0/0/0       ## section ike-phase1, proposal, policy, gateway
    gateway: 172.17.10.2
    localAddress: 172.16.10.1
    authenticationAlgorithm: sha-256
    encryptionAlgorithm: aes-256-cbc
    preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    trafficSelector:
    - name: ts1
      localIP: 10.111.1.0/24 ## IP cannot be 0.0.0.0/0
      remoteIP: 10.221.1.0/24 ## IP cannot be 0.0.0.0/0
  - interface: ge-0/0/1       ## section ike-phase1, proposal, policy, gateway
    gateway: 172.18.10.2
    localAddress: 172.16.10.1
    authenticationAlgorithm: sha-256
    encryptionAlgorithm: aes-256-cbc
    preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    trafficSelector:
    - name: ts2
      localIP: 10.112.1.0/24 ## IP cannot be 0.0.0.0/0
      remoteIP: 10.222.1.0/24 ## IP cannot be 0.0.0.0/0

jcnr_config:
  - name: ge-0/0/0
```

```
    routes:
      - "10.221.1.0/24"
      - "10.222.1.0/24"
```

Configuring cSRX IPSec Tunnel Configuration via Configlets

You can configure the IPSec tunnels after cSRX has been deployed using a configlet. The installation helm chart must have `enableUserConfig` flag value set to true. The `ipSecTunnelConfigs` configuration snippet must not be defined. Sample helm charts are provided in the installer bundle— `/csrx_examples/values-user-config-csrx.yaml` for cSRX only installation and `/charts/junos-csrx/csrx_examples/values-user-config-unified.yaml` for unified JCNR and cSRX installation. An example helm chart is provided below:

```
enableUserConfig: true  # enable /disable user configuration

interfaceType: "vhost"

interfaceConfigs:
    - name: ge-0/0/0
      ip: 172.16.10.1/30          # should match ipSecTunnelConfigs localAddress if configured
      gateway: 172.16.10.2        # gateway configuration
      ip6: 2001:db8:172:16:10::1/64        # optional
      ip6Gateway: 2001:db8:172:16:10::2    # optional
      routes:                   # this field is optional
      - "172.17.10.0/24"
      - "172.18.10.0/24"
      instance_parameters:
        name: "untrust"
        type: "vrf"        # options include virtual-router or vrf
        vrfTarget: 10:10
    - name: ge-0/0/1
      ip: 192.168.1.1/30           # should match ipSecTunnelConfigs localAddress if configured
      gateway: 192.168.1.2         # gateway configuration
      ip6: 2001:db8:192:168:1::1/64                    # optional
      ip6Gateway: 2001:db8:192:168:1::2              # optional
      routes:                   # this field is optional
      - "10.111.1.0/24"
      - "10.112.1.0/24"
      instance_parameters:
        name: "trust"
        type: "vrf"        # options include virtual-router or vrf
        vrfTarget: 11:11
```

```
  jcnr_config:
    - name: ge-0/0/0
      routes:
        - "10.221.1.0/24"
        - "10.222.1.0/24"
```

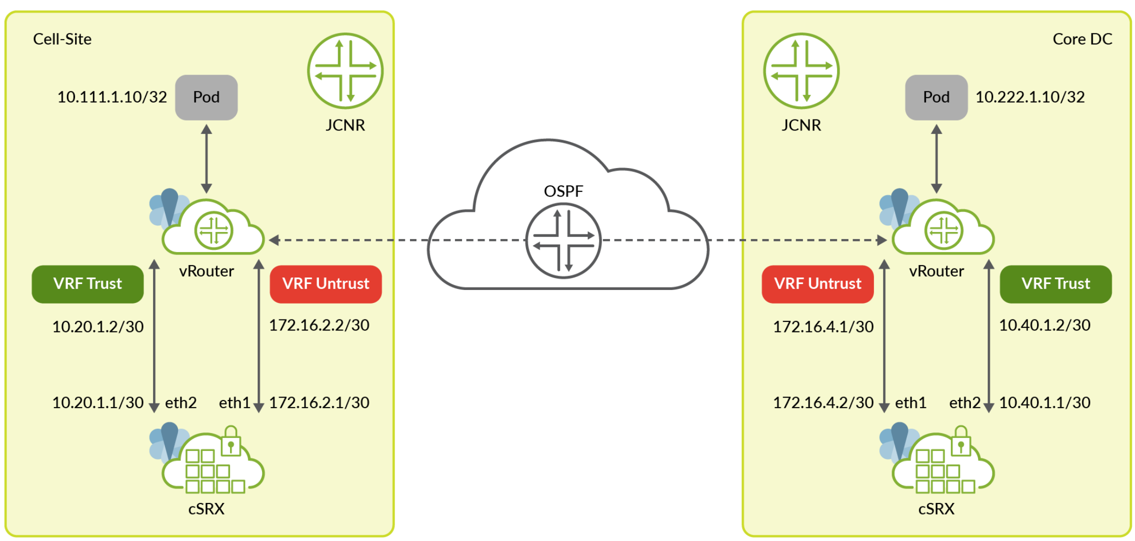
You can create IPSec tunnels or modify tunnel configuration after deployment using a configlet. A sample configlet has been provided in the installer bundle—`/csrx_examples/ipsec-tunnel-config-cr.yaml` for cSRX only installation and `/charts/junos-csrx/csrx_examples/ipsec-tunnel-config-cr.yaml` for unified JCNR and cSRX installation. An example configlet is provided below:

```
# Example configlet CR to configure an IPsec tunnel on a CSRX
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: example-csrx-cr
  namespace: jcnr
  labels:
    app: csrx
  annotations:
    juniper.net/device: csrx
spec:
  config: |-
    set security ike proposal ike-phase-171-1-1-1-0-proposal dh-group group5
    set security ike proposal ike-phase-171-1-1-1-0-proposal encryption-algorithm aes-256-cbc
    set security ike proposal ike-phase-171-1-1-1-0-proposal authentication-algorithm sha-256
    set security ike proposal ike-phase-171-1-1-1-0-proposal authentication-method pre-shared-
keys
    set security ike proposal ike-phase-171-1-1-1-0-proposal lifetime-seconds 3600
    set security ike policy ike-phase-171-1-1-1-0-policy proposals ike-phase-171-1-1-1-0-proposal
    set security ike policy ike-phase-171-1-1-1-0-policy pre-shared-key ascii-text
"$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    set security ike gateway remote-171-1-1-1-0 ike-policy ike-phase-171-1-1-1-0-policy
    set security ike gateway remote-171-1-1-1-0 address 181.1.1.1
    set security ike gateway remote-171-1-1-1-0 external-interface ge-0/0/0.0
    set security ike gateway remote-171-1-1-1-0 local-address 171.1.1.1
    set security ike gateway remote-171-1-1-1-0 version v2-only
    set security ipsec proposal ipsec-171-1-1-1-0-proposal protocol esp
    set security ipsec proposal ipsec-171-1-1-1-0-proposal encryption-algorithm aes-256-cbc
    set security ipsec proposal ipsec-171-1-1-1-0-proposal authentication-algorithm hmac-
sha-256-128
```

```
set security ipsec proposal ipsec-171-1-1-1-0-proposal lifetime-seconds 18000
set security ipsec policy ipsec-171-1-1-1-0-policy perfect-forward-secrecy keys group5
set security ipsec policy ipsec-171-1-1-1-0-policy proposals ipsec-171-1-1-1-0-proposal
set security ipsec vpn ipsec-remote-171-1-1-1-0 bind-interface st0.0
set security ipsec vpn ipsec-remote-171-1-1-1-0 ike gateway remote-171-1-1-1-0
set security ipsec vpn ipsec-remote-171-1-1-1-0 ike ipsec-policy ipsec-171-1-1-1-0-policy
set security ipsec vpn ipsec-remote-171-1-1-1-0 establish-tunnels immediately
set security ipsec vpn ipsec-remote-171-1-1-1-0 traffic-selector ts1 local-ip 100.1.1.0/24
set security ipsec vpn ipsec-remote-171-1-1-1-0 traffic-selector ts1 remote-ip 200.1.1.0/24
set security zones security-zone untrust host-inbound-traffic system-services all
set security zones security-zone untrust host-inbound-traffic protocols all
```

Configuration Example

Let us look at a configuration example for Cloud-Native Router with security services. Consider the following network topology:



The topology consists of a Cloud-Native Router on the cell-site and a Cloud-Native Router in the core data center. Both JCNRs are service chained with cSRX. The core data center can also have any other physical or virtual firewall function as the tunnel endpoint. The traffic between pods `10.111.1.10` and `10.222.1.10` must be encrypted through an IPsec tunnel. The IP addresses and gateway addresses for the cSRX-Cloud-Native Router interface are also illustrated.

cSRX connects with JCNR's forwarding plane (vRouter) using two interfaces:

- ge-0/0/1 interface is used to send traffic from Cloud-Native Router to security services for IPsec encryption and from security services to Cloud-Native Router after IPsec decryption. This interface is a part of the trust zone in cSRX and trust VRF in JCNR.
- ge-0/0/0 interface is used to send traffic from security services to Cloud-Native Router after IPsec encryption and from Cloud-Native Router to security services for IPsec decryption. This interface is part of the untrust zone in cSRX and untrust VRF in JCNR.

Let us look at the configuration steps:

1. Configure the cSRX helm chart with correct `interfaceConfigs`, `ipSecTunnelConfigs` and `jcnr_config` for the topology.

   a. Helm chart configuration for cell-site JCNR:

```
junos-csrx:
  # Default values for cSRX.
  # This is a YAML-formatted file.
  # Declare variables to be passed into your templates.

  common:
    registry: enterprise-hub.juniper.net/
    repository: jcnr-container-prod/

    csrxInit:
      repository:
      image: csrx-init
      tag: R25.1-25
      imagePullPolicy: IfNotPresent
      resources:
        #limits:
        #   memory: 1Gi
        #   cpu: 1
        #requests:
        #   memory: 1Gi
        #   cpu: 1

    csrx:
      repository:
      image: csrx
      tag: 25.1R1.8
      imagePullPolicy: IfNotPresent
      resources:
        limits:
```

```
          hugepages-1Gi: 6Gi
          memory: 4Gi
        requests:
          hugepages-1Gi: 6Gi
          memory: 4Gi

    csrxTelemetry:
      repository:
      image: contrail-telemetry-exporter
      tag: 25.1.0.25
      imagePullPolicy: IfNotPresent
      resources:


  # kubeconfigpath: path to the kubeconfig file (to override the default path /etc/
kubernetes/kubelet.conf)
  # kubeConfigPath: /path/to/kubeconfig

  # nodeAffinity: Can be used to inject nodeAffinity for cSRX
  # you may label the nodes where we wish to deploy cSRX and inject affinity accordingly
  nodeAffinity:
  #- key: node-role.kubernetes.io/worker
  #  operator: Exists
  #- key: node-role.kubernetes.io/master
  #  operator: DoesNotExist
  - key: kubernetes.io/hostname
    operator: In
    values:
    - node2

  replicas: 1

  interfaceType: "vhost"

interfaceConfigs:
  - name: ge-0/0/0
    ip: 172.16.2.1/30          # --> Interface IP in Untrust VRF, should match
ipSecTunnelConfigs localAddress if configured
    gateway: 172.16.2.2        # --> gateway configuration
    ip6: 2001:172:16:2::1/126        # optional
    ip6Gateway: 2001:172.16.2::2    # optional
    routes:                   # --> Route to remote tunnel endpoint
    - "172.16.4.0/24"
```

```
    instance_parameters:
       name: "untrust"
       type: "vrf"        # options include virtual-router or vrf
       vrfTarget: 10:10
  - name: ge-0/0/1
    ip: 10.20.1.1/30           # --> Interface IP in the Trust VRF
    gateway: 10.20.1.2         # --> gateway configuration
    ip6: 2001:10:20:1::1/126           # optional
    ip6Gateway: 2001:10:20:1::2        # optional
    routes:                   # --> Route to local application subnet
    - "10.111.1.0/24"
    instance_parameters:
       name: "trust"
       type: "vrf"        # options include virtual-router or vrf
       vrfTarget: 11:11

ipSecTunnelConfigs:       # untrust
  - interface: ge-0/0/0       ## section ike-phase1, proposal, policy, gateway
    gateway: 172.16.4.1   # --> Remote tunnel endpoint
    localAddress: 172.16.2.1  # --> Local Untrust Interface IP
    authenticationAlgorithm: sha-256
    encryptionAlgorithm: aes-256-cbc
    preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    trafficSelector:
    - name: ts1
      localIP: 10.111.1.0/24  # --> Traffic selector based on local application subnet
      remoteIP: 10.222.1.0/24 # --> Traffic selector based on remote application subnet

jcnr_config:
  - name: ge-0/0/1
    routes:
      - "10.222.1.0/24"     # --> cRPD route to remote application subnet via trust
interface

#csrx_flavor: specify the csrx deployment model. Corresponding values for csrx control and
data cpus
#must be provided based on the flavor mentioned below. Following are possible options:
# CSRX-2CPU-4G
# CSRX-4CPU-8G
# CSRX-6CPU-12G
# CSRX-8CPU-16G
# CSRX-16CPU-32G
# CSRX-20CPU-48G
```

```
csrx_flavor: CSRX-2CPU-4G

csrx_ctrl_cpu: "0x01"

csrx_data_cpu: "0x02"
```

b. Helm chart configuration for remote JCNR:

```
# Default values for cSRX.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

common:
  registry: enterprise-hub.juniper.net/
  repository: jcnr-container-prod/

  csrxInit:
    image: csrx-init
    tag: f4tgt33
    imagePullPolicy: IfNotPresent
    resources:
      #limits:
      #  memory: 1Gi
      #  cpu: 1
      #requests:
      #  memory: 1Gi
      #  cpu: 1

  csrx:
    image: csrx
    tag: 24.2R1.14
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        hugepages-1Gi: 4Gi
        memory: 4Gi
      requests:
        hugepages-1Gi: 4Gi
        memory: 4Gi

# uncomment below if you are using a private registry that needs authentication
# registryCredentials - Base64 representation of your Docker registry credentials
```

```
# secretName - Name of the Secret object that will be created
#imagePullSecret:
  #registryCredentials:  <base64-encoded-credential>
  #secretName: regcred

# nodeAffinity: Can be used to inject nodeAffinity for cSRX
# you may label the nodes where we wish to deploy cSRX and inject affinity accordingly
# nodeAffinity:
#- key: node-role.kubernetes.io/worker
#  operator: Exists
#- key: node-role.kubernetes.io/master
#  operator: DoesNotExist

replicas: 1

interfaceType: "vhost"

interfaceConfigs:
  - name: ge-0/0/0
    ip: 172.16.4.1/30          # --> Interface IP in Untrust VRF, should match
ipSecTunnelConfigs localAddress if configured
    gateway: 172.16.4.2        # --> gateway configuration
    ip6: 2001:172:16:4::1/126        # optional
    ip6Gateway: 2001:172.16.4::2    # optional
    routes:                    # --> Route to remote tunnel endpoint
    - "172.16.2.0/24"
    instance_parameters:
        name: "untrust"
        type: "vrf"        # options include virtual-router or vrf
        vrfTarget: 10:10
  - name: ge-0/0/1
    ip: 10.40.1.1/30           # --> Interface IP in the Trust VRF
    gateway: 10.40.1.2         # --> gateway configuration
    ip6: 2001:10:40:1::1/126           # optional
    ip6Gateway: 2001:10:40:1::2        # optional
    routes:                    # --> Route to local application subnet
    - "10.222.1.0/24"
    instance_parameters:
       name: "trust"
       type: "vrf"        # options include virtual-router or vrf
       vrfTarget: 11:11

ipSecTunnelConfigs:       # untrust
```

```
  - interface: ge-0/0/0       ## section ike-phase1, proposal, policy, gateway
    gateway: 172.16.2.1   # --> Remote tunnel endpoint
    localAddress: 172.16.4.1  # --> Local Untrust Interface IP
    authenticationAlgorithm: sha-256
    encryptionAlgorithm: aes-256-cbc
    preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    trafficSelector:
    - name: ts1
      localIP: 10.222.1.0/24  # --> Traffic selector based on local application subnet
      remoteIP: 10.111.1.0/24 # --> Traffic selector based on remote application subnet

jcnr_config:
  - name: ge-0/0/0
    routes:
      - "10.222.1.0/24"      # --> cRPD route to local application subnet via untrust
interface
  - name: ge-0/0/1
    routes:
      - "10.111.1.0/24"     # --> cRPD route to remote application subnet via trust
interface

#csrx_flavor: specify the csrx deployment model. Corresponding values for csrx control and
data cpus
#must be provided based on the flavor mentioned below. Following are possible options:
# CSRX-2CPU-4G
# CSRX-4CPU-8G
# CSRX-6CPU-12G
# CSRX-8CPU-16G
# CSRX-16CPU-32G
# CSRX-20CPU-48G
csrx_flavor: CSRX-2CPU-4G

csrx_ctrl_cpu: "0x01"

csrx_data_cpu: "0x02"
```

Once you have configured the helm chart, you must deploy cSRX.

Please review the *Deploying Service Chain (cSRX) with JCNR* topic for details on how to deploy cSRX for service chaining with JCNR.

2. Configure the Cloud-Native Router fabric interface (`ens192`) to participate in the IGP running in the core. The configuration is performed in the untrust VRF.

a. Example configlet for OSPF on the cell-site JCNR:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-ipsec-ospf             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set policy-options policy-statement export_static_ospf from protocol local
    set policy-options policy-statement export_static_ospf from protocol static
    set policy-options policy-statement export_static_ospf then accept
    set routing-instances untrust protocols ospf export export_static_ospf
    set routing-instances untrust protocols ospf area 0 interface ens192
    set interfaces ens192 unit 0 family inet address 172.16.0.11/24
    set routing-instances untrust interface ens192
  crpdSelector:
    matchLabels:
        node: worker
```

b. Example configlet for OSPF on the remote JCNR:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-ipsec-ospf             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set policy-options policy-statement export_static_ospf from protocol local
    set policy-options policy-statement export_static_ospf from protocol static
    set policy-options policy-statement export_static_ospf then accept
    set routing-instances untrust protocols ospf export export_static_ospf
    set routing-instances untrust protocols ospf area 0 interface ens192
    set interfaces ens192 unit 0 family inet address 172.16.0.12/24
    set routing-instances untrust interface ens192
  crpdSelector:
    matchLabels:
        node: worker
```

3. Deploy the application pods with an interface attached to the trust VRF in JCNR.

a. Application pod on the cell-site:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-trust
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "net-trust",
    "plugins": [
      {
        "type": "jcnr",               # --> CNI plugin is jcnr
        "args": {
          "vrfName": "trust",         # --> VRF name is trust
          "vrfTarget": "10:10"
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
---
apiVersion: v1
kind: Pod
metadata:
  name: pktgen-ce1
  labels:
     app: pktgen-odu
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "net-trust",
          "interface":"net1",
          "cni-args": {
            "interfaceType":"veth",
            "mac":"aa:bb:cc:dd:50:11",
            "ipConfig":{
              "ipv4":{
                "address":"10.111.1.10/24",      # --> IP address of the pod
                "gateway":"10.111.1.1",
                "routes":[ "10.222.1.0/24"]
```

```
                },
                "ipv6":{
                        "address":"2001:0db8:10:111:1::10/126",
                        "gateway":"2001:0db8:10:111:1::10",
                        "routes":["2001:0db8:10:222:1::0/126"]
                }
              }
            }
          }
        ]
spec:
  affinity:
<trimmed...>
```

b. Application pod on the remote site:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-trust
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "net-trust",
    "plugins": [
      {
        "type": "jcnr",               # --> CNI plugin is jcnr
        "args": {
          "vrfName": "trust",         # --> VRF name is trust
          "vrfTarget": "10:10"
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
---
apiVersion: v1
kind: Pod
metadata:
  name: pktgen-ce1
  labels:
     app: pktgen-odu
```

```
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "net-trust",
          "interface":"net1",
          "cni-args": {
            "interfaceType":"veth",
            "mac":"aa:bb:cc:dd:50:22",
            "ipConfig":{
              "ipv4":{
                "address":"10.222.1.10/24",      # --> IP address of the pod
                "gateway":"10.222.1.1",
                "routes":[ "10.111.1.0/24"]
              },
              "ipv6":{
                      "address":"2001:0db8:10:222:1::10/126",
                      "gateway":"2001:0db8:10:222:1::10",
                      "routes":["2001:0db8:10:111:1::0/126"]
              }
            }
          }
        }
      ]
spec:
  affinity:
<trimmed...>
```

Verify Configuration

You can verify the configuration and traffic flows in cRPD, cSRX and vRouter.

1. Verify cRPD configuration for trust and untrust VRFs via the "cRPD shell" on page 329. The configuration is available under the cni configuration group.

```
user@host > show configuration groups cni | display set
set groups cni apply-flags omit
set groups cni apply-macro ht jcnr
set groups cni routing-instances untrust instance-type vrf
set groups cni routing-instances untrust routing-options rib untrust.inet6.0 static route
```

```
2001:db8:172:16:10::1/128 qualified-next-hop 2001:db8:172:16:10::1 interface
vhostge-0_0_0-90aca656-8b86-4d1d-a8
set groups cni routing-instances untrust routing-options static route 172.16.10.1/32
qualified-next-hop 172.16.10.1 interface vhostge-0_0_0-90aca656-8b86-4d1d-a8
set groups cni routing-instances untrust interface vhostge-0_0_0-90aca656-8b86-4d1d-a8
set groups cni routing-instances untrust route-distinguisher 10:10
set groups cni routing-instances untrust vrf-target target:10:10
set groups cni routing-instances trust instance-type vrf
set groups cni routing-instances trust routing-options rib trust.inet6.0 static route
2001:db8:10:20:1::1/128 qualified-next-hop 2001:db8:10:20:1::1 interface
vhostge-0_0_1-90aca656-8b86-4d1d-a8
set groups cni routing-instances trust routing-options static route 10:20:1.1/32 qualified-
next-hop 10:20:1.1 interface vhostge-0_0_1-90aca656-8b86-4d1d-a8
set groups cni routing-instances trust routing-options static route 10.222.1.0/24 qualified-
next-hop 10.20.1.1 interface vhostge-0_0_1-90aca656-8b86-4d1d-a8
set groups cni routing-instances trust interface vhostge-0_0_1-90aca656-8b86-4d1d-a8
set groups cni routing-instances trust route-distinguisher 11:11
set groups cni routing-instances trust vrf-target target:11:11
```

2. Verify the cRPD Routing Tables:

```
user@host > show route table trust.inet.0
trust.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.20.1.1/32       *[Static/5] 03:59:00
                    >  via vhostge-0_0_1-a6c764da-a46d-4832-b5
10.20.1.2/32       *[Local/0] 03:59:00
                       Local via vhostge-0_0_1-a6c764da-a46d-4832-b5
10.111.1.1/32      *[Local/0] 03:59:00
                       Local via jvknet1-88e1126
10.111.1.10/32     *[Static/5] 03:59:00
                    >  via jvknet1-88e1126
10.222.1.0/24      *[Static/5] 03:59:00
                    >  via vhostge-0_0_1-a6c764da-a46d-4832-b5
```

```
user@host > show route table untrust.inet.0
untrust.inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
10.111.1.0/24      *[Static/5] 04:00:31
                    >  via vhostge-0_0_0-a6c764da-a46d-4832-b5
10.222.1.0/24      *[OSPF/150] 03:56:03, metric 0, tag 0
                    >  to 172.16.0.12 via ens192
172.16.0.0/24      *[Direct/0] 04:00:31
                    >  via ens192
172.16.0.11/32     *[Local/0] 04:00:31
                       Local via ens192
172.16.1.0/24      *[OSPF/10] 03:56:03, metric 2
                    >  to 172.16.0.12 via ens192
172.16.2.1/32      *[Static/5] 04:00:31
                    >  via vhostge-0_0_0-a6c764da-a46d-4832-b5
172.16.2.2/32      *[Local/0] 04:00:31
                       Local via vhostge-0_0_0-a6c764da-a46d-4832-b5
172.16.4.1/32      *[OSPF/150] 03:56:03, metric 0, tag 0
                    >  to 172.16.0.12 via ens192
```

3. Login to the cSRX shell using the `kubectl exec -it` *`csrx_pod_name`* `-n jcnr -- bash` command. Type `cli` to navigate to the CLI mode. Verify the cSRX configuration, security associations and flows:

```
user@host> show configuration
## Last commit: 2024-10-01 10:25:50 UTC by root
version 20240621.103832_builder.r1429411;
system {
    root-authentication {
        encrypted-password *disabled*; ## SECRET-DATA
    }
    services {
        ssh {
            root-login allow;
        }
    }
}
interfaces {
    ge-0/0/0 {
        unit 0 {
            family inet {
                address 172.16.2.1/30;
            }
        }
    }
    ge-0/0/1 {
```

```
        unit 0 {
            family inet {
                address 10.20.1.1/30;
            }
        }
    }
    st0 {
        unit 0 {
            family inet;
        }
    }
}
routing-options {
    static {
        route 172.16.4.0/24 next-hop 172.16.2.2/32;
        route 10.111.1.0/24 next-hop 10.20.1.2/32;
    }
}
security {
    ike {
        proposal ike-phase-172-16-2-1-proposal {
            authentication-method pre-shared-keys;
            dh-group group5;
            authentication-algorithm sha-256;
            encryption-algorithm aes-256-cbc;
            lifetime-seconds 3600;
        }
        policy ike-phase-172-16-2-1-policy {
            proposals ike-phase-172-16-2-1-proposal;
            pre-shared-key ascii-text
"$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"; ## SECRET-DATA
        }
        gateway remote-172-16-2-1 {
            ike-policy ike-phase-172-16-2-1-policy;
            address 172.16.4.1;
            external-interface ge-0/0/0.0;
            local-address 172.16.2.1;
            version v2-only;
        }
    }
    ipsec {
        proposal ipsec-172-16-2-1-proposal {
            protocol esp;
```

```
        }
        policy ipsec-172-16-2-1-policy {
            perfect-forward-secrecy {
                keys group5;
            }
            proposals ipsec-172-16-2-1-proposal;
        }
        vpn ipsec-remote-172-16-2-1 {
            bind-interface st0.0;
            ike {
                gateway remote-172-16-2-1;
                ipsec-policy ipsec-172-16-2-1-policy;
            }
            traffic-selector ts1 {
                local-ip 10.111.1.0/24;
                remote-ip 10.222.1.0/24;
            }
            establish-tunnels immediately;
        }
    }
    policies {
        default-policy {
            permit-all;
        }
    }
    zones {
        security-zone untrust {
            host-inbound-traffic {
                system-services {
                    all;
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/0.0;
                st0.0;
            }
        }
        security-zone trust {
            host-inbound-traffic {
                system-services {
```

```
                    all;
                }
                protocols {
                    all;
                }
            }
            interfaces {
                ge-0/0/1.0;
            }
        }
    }
}
```

```
user@host> show security ike security-associations
Index   State  Initiator cookie  Responder cookie  Mode           Remote Address
73      UP     6895337b7ae4b449  2bdca7788a896e50  IKEv2          172.16.4.1
```

```
user@host> show security ipsec security-associations
  Total active tunnels: 1     Total IPsec sas: 1
  ID      Algorithm       SPI      Life:sec/kb  Mon lsys Port  Gateway
  <500002 ESP:3des/sha1   0x0283e69b 2132/ unlim -  root 500   172.16.4.1
  >500002 ESP:3des/sha1   0xbdd5cc1e 2132/ unlim -  root 500   172.16.4.1
```

```
user@host> show security ipsec statistics
ESP Statistics:
  Encrypted bytes:          2878576
  Decrypted bytes:          1739472
  Encrypted packets:          21166
  Decrypted packets:          20708
AH Statistics:
  Input bytes:                    0
  Output bytes:                   0
  Input packets:                  0
  Output packets:                 0
Errors:
  AH authentication failures: 0, Replay errors: 0
  ESP authentication failures: 0, ESP decryption failures: 0
  Bad headers: 0, Bad trailers: 0
  Invalid SPI: 0, TS check fail: 0
```

```
  Exceeds tunnel MTU: 0
  Discarded: 0
```

```
user@host> show security flow session
Session ID: 2, Policy name: N/A, Timeout: N/A, Session State: Valid
  In: 172.16.4.1/0 --> 172.16.2.1/0;esp, Conn Tag: 0x0, If: ge-0/0/0.0, Pkts: 0, Bytes: 0,

Session ID: 2696, Policy name: N/A, Timeout: N/A, Session State: Valid
  In: 172.16.4.1/20915 --> 172.16.2.1/10740;esp, Conn Tag: 0x0, If: ge-0/0/0.0, Pkts: 7,
Bytes: 952,
Total sessions: 2
```

4. You can also verify the flow in the "vRouter CLI" on page 331:

```
# flow -l
Flow table(size 161218560, entries 629760)
…
    Index                Source:Port/Destination:Port                     Proto(V)
-----------------------------------------------------------------------------------
    58256<=>206532       172.16.4.1:0                                      50 (1)
                         172.16.2.1:0
(Gen: 53, K(nh):1, Action:F, Flags:, QOS:-1, S(nh):26,  Stats:14479/1969144,
 SPort 63031, TTL 0, Sinfo 0.0.0.0)

   202824<=>382336       10.111.1.10:67                                     1 (2)
                         10.222.1.10:0
(Gen: 5, K(nh):2, Action:F, Flags:, QOS:-1, S(nh):18,  Stats:14754/1445892,
 SPort 51876, TTL 0, Sinfo 8.0.0.0)

   206532<=>58256        172.16.2.1:0                                      50 (1)
                         172.16.4.1:0
(Gen: 53, K(nh):1, Action:F, Flags:, QOS:-1, S(nh):27,  Stats:14494/2174100,
 SPort 55303, TTL 0, Sinfo 6.0.0.0)

   382336<=>202824       10.222.1.10:67                                     1 (2)
                         10.111.1.10:0
(Gen: 5, K(nh):2, Action:F, Flags:, QOS:-1, S(nh):14,  Stats:14479/1418942,
 …
```

RELATED DOCUMENTATION

IPsec Overview

Cloud-Native Router as a Transit Gateway

Cloud-Native Router can act as a transit gateway for external traffic. As a transit gateway, Cloud-Native Router is neither the source nor the destination for the traffic, but an intermediate hop. It acts as a vanilla router to switch traffic between multiple physical interfaces.

Starting with Juniper Cloud-Native Router (JCNR) Release 23.2, Cloud-Native Router can now act as a transit gateway for external traffic. As a transit gateway, Cloud-Native Router is neither the source nor the destination for the traffic, but an intermediate hop. It acts as a vanilla router to switch traffic between multiple physical interfaces. Depending on the forwarding state, Cloud-Native Router can encapsulate or decapsulate the traffic between interfaces.

> **NOTE**: Starting with Cloud-Native Router Release 23.2, Cloud-Native Router supports multiple fabric interfaces that enable it to function as a transit gateway.

Cloud-Native Router has to be deployed in the L3 mode to perform the transit router functionality. Add all physical interfaces (physical and virtual functions) as fabric interfaces in the helm chart before deploying the JCNR. The deployed Cloud-Native Router does not support editing or changing the fabric interfaces during run time. However, you can create or remove pod interfaces during run time. Here are example helm chart configurations:

```
fabricInterface:
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"
```

```
fabricInterface:
   - subnet: 10.0.3.0/24
     gateway: 10.0.3.1
     ddp: "off"
   - subnet: 10.0.5.0/24
     gateway: 10.0.5.1
     ddp: "off"
```

You need to configure an IP address on the loopback interface and use it as a tunnel endpoint for each Cloud-Native Router instance. The loopback IP address is the next hop address which BGP advertises to its peers. All data packets with encapsulations like MPLSoUDP will have the outer IP address as this loopback IP address. The loopback IP address is reachable via any of the physical interfaces. The loopback IP address should be in a /32 subnet without a MAC address. For example:

```
set interfaces lo1 unit 1 family inet address 10.0.0.1/32
```

EVPN Type 5 Routing over VXLAN Tunnels

IN THIS SECTION

Ethernet Virtual Private Network (EVPN) with Virtual Extensible LAN (VXLAN) Type 5 routing is designed for use in data center and cloud environments to provide efficient and scalable network connectivity for virtualized workloads. It combines the benefits of EVPN and VXLAN to enable flexible and seamless communication between virtual machines (VMs) and physical devices across different IP subnets and locations. Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, Cloud-Native Router supports EVPN Type 5 Routing over VXLAN tunnels.

Ethernet Virtual Private Network (EVPN) technology provides a scalable and efficient way to extend Layer 2 and Layer 3 connectivity across multiple sites. EVPN uses Border Gateway Protocol (BGP) to exchange information between Provider Edge (PE) routers, allowing them to learn the location of Ethernet segments and IP prefixes. This allows for the creation of virtual networks that can span multiple sites, while providing traffic separation and isolation through the use of virtual routing and forwarding (VRF) instances. EVPN supports several encapsulation methods, including VXLAN and MPLS, which can be used to transport traffic across the service provider network.

VXLAN is a network overlay technology that allows the creation of virtual Layer 2 networks on top of an existing Layer 3 network infrastructure. It extends the reach of Layer 2 segments beyond the confines of a single physical network, which is especially useful in large-scale virtualized environments.

EVPN supports two types of routes: MAC Advertisement Route (Type 2) and IP Prefix Route (Type 5). Type 2 routes are used to exchange MAC addresses and VLANs between PE routers, while Type 5 routes are used to exchange Layer 3 network routes. In EVPN VXLAN, Type 5 routes are used to advertise IP prefixes and their associated MAC addresses. To reach a tenant using connectivity provided by the EVPN VXLAN Type 5 IP prefix route, data packets are sent as Layer 2 Ethernet frames encapsulated in the VXLAN header over the IP network across the data centers.

EVPN VXLAN Type 5 routing allows for efficient distribution of MAC and IP routing information, enabling large-scale networks with numerous virtualized workloads to operate seamlessly. The technology supports secure isolation of tenant traffic in shared environments, providing a virtual network overlay that maintains separation between tenants.

To learn more about EVPN VXLAN Type 5 routing, see *EVPN VXLAN Type 5 Routing* topic.

NOTE: Transit router functionality should be enabled for Cloud-Native Router to support EVPN VXLAN Type 5 routing. See, "Cloud-Native Router as a Transit Gateway" on page 190.

Enabling EVPN Type 5 Routing over VXLAN Tunnels

NOTE: Use the *configlet resource* to configure the cRPD pods.

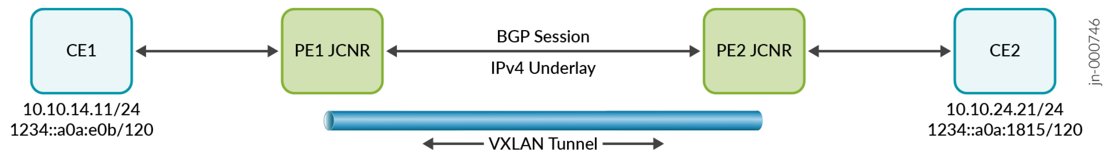
An example EVPN VXLAN Type 5 configuration snippet is provided below:

```
routing-instances {
    EVPN-TYPE5-VXLAN-VRF {
        instance-type vrf;
            protocols {
                evpn {
                    ip-prefix-routes {
                        advertise direct-nexthop;
                        encapsulation vxlan;
                        vni 1000;
                        export EVPN-TYPE5-VXLAN-VRF-EXPORT-POLICY;
                    }
                }
            }
            interface ge-0/0/1.0;
            route-distinguisher 10.255.0.1:100;
            vrf-target target:100:100;
        }
```

```
    }
}
```

To learn about EVPN Type 5 configuration in Junos, see *EVPN Type 5 Configuration* topic.

Configuration Example and CLI Commands for EVPN Type 5 Routing over VXLAN Setup



The topology shown above describes a simple setup with two JCNRs deployed as provider edge routers PE1 and PE2. The CE1 and CE2 represent hosts behind each of the PEs. As a pre-requisite, a BGP session must exist between PE1 and PE2.

NOTE: Use the *configlet resource* to configure the cRPD pods.

Consider the following EVPN-VXLAN configuration on PE1, with the interface `enp4s0` towards CE1:

```
routing-instances {
    orange {
        instance-type vrf;
        routing-options {
            rib orange.inet6.0 {
                multipath;
            }
            multipath;
        }
        protocols {
            evpn {
                ip-prefix-routes {
                     advertise direct-nexthop;
                     encapsulation vxlan;
                     vni 10010;
                }
```

```
            }
        }
        interface enp4s0;
        route-distinguisher 1.1.1.1:4;
        vrf-target target:4:4;
    }
}
```

A VXLAN tunnel is created between routers PE1 and PE2. The 10.10.14.0/24 network routes are locally learnt on PE1 and are advertised via EVPN Type 5 to the remote PE. Similarly, the 10.10.24.0/24 network routes are locally learnt on PE2 and advertised via EVPN Type 5 to the remote PE. All traffic between CE1 and CE2 is forwarded between PE1 and PE2 over the VXLAN tunnel.

Use the commands listed in the sections below to troubleshoot a EVPN VXLAN Type 5 routing setup.

cRPD CLI Commands

The following CLI commands can be executed on the cRPD CLI. To access the cRPD CLI, see "Access cRPD CLI" on page 329.

- `show bgp <summary | neighbor>`: Provides a summary of the EVPN connection to the peer and the status of the connection.

  A sample output is shown below:

  ```
  host@pe1> show bgp summary
  Threading mode: BGP I/O
  Default eBGP mode: advertise - accept, receive - accept
  Groups: 1 Peers: 2 Down peers: 1
  Table                Tot Paths  Act Paths  Suppressed  History Damp  State     Pending
  bgp. evpn. 0         2          2          0           0             0         0

  Peer                          AS     InPkt     OutPkt     OutQ  Flaps   Last   Up/Dwn
  State|#Active/Received/Accepted/Damped…
  2.2.2.2                    4      10345    10336     0      2      3d     5:32:50
  Establ
  bgp.evpn.0: 2/2/2/0
  orange.evpn.0: 2/2/2/0
  3.3.3.3                       4      0       0          0      0      4w4d   13:28:22
  Connect
  ```

- `show route <summary | table | prefix>`: Displays the active entries in the routing tables.

- `show evpn instance`: Displays information about the EVPN routing instance.
- `show evpn l3-context`: Displays the configured L3 context on the local box.

  A sample output is shown below:

```
host@pe1> show evpn l3-context
L3 context                      Type  Adv      Encap  VNI/Label  Router MAC/GW intf
orange                          Cfg   Direct   VXLAN  10010      48:5a:0d:78:78:d7
```

- `show evpn ip-prefix-database`: Provides a list of exported and imported EVPN route prefixes and the status of these routes.

  A sample output is shown below:

```
root@evpn-pe1-node> show evpn ip-prefix-database
L3 context: orange

IPv4->EVPN Exported Prefixes
Prefix                                       EVPN route status
2.55.1.0/24                                  Created
4.1.1.4/30                                   Created
10.10.14.0/24                                Created

IPv6->EVPN Exported Prefixes
Prefix                                       EVPN route status
1234::a0a:e00/120                            Created
abcd::401:104/126                            Created
abcd::2:55:1:0/120                           Created

EVPN->IPv4 Imported Prefixes
Prefix                                       Etag
2.55.2.0/24                                  0
  Route distinguisher    VNI/Label  Router MAC         Nexthop/Overlay GW/ESI   Route-Status
Reject-Reason
  2.2.2.2:4              10020      48:5a:0d:49:fc:63  2.2.2.2
Accepted      n/a
10.10.24.0/24                                0
  Route distinguisher    VNI/Label  Router MAC         Nexthop/Overlay GW/ESI   Route-Status
Reject-Reason
  2.2.2.2:4              10020      48:5a:0d:49:fc:63  2.2.2.2
Accepted      n/a
```

```
EVPN->IPv6 Imported Prefixes
Prefix                                      Etag
1234::a0a:1800/120                          0
  Route distinguisher    VNI/Label  Router MAC         Nexthop/Overlay GW/ESI   Route-Status
Reject-Reason
  2.2.2.2:4              10020      48:5a:0d:49:fc:63  2.2.2.2
Accepted      n/a
abcd::2:55:2:0/120                          0
  Route distinguisher    VNI/Label  Router MAC         Nexthop/Overlay GW/ESI   Route-Status
Reject-Reason
  2.2.2.2:4              10020      48:5a:0d:49:fc:63  2.2.2.2
Accepted      n/a
```

- `show route table <VRF>.evpn.0`: Displays the route entries in the specified routing table.

  A sample output is shown below.

```
host@pe1> show route table orange.evpn.0

orange.evpn.0: 4 destinations, 0 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route,  - = Last Active,  * = Both

5:1.1.1.1:4::0::10.10.14.0::24/248
                            *[EVPN/170] 4W4d 13:29:25
                                Fictitious
5:2.2.2.2:4::0::10.10.24.0::24/248
                              *[BGP/170] 3d 05:33:52, localpref 100, from 2.2.2.2
                                    AS path: I, validation-state: unverified
                                     to 10.10.1.20 via enp2s0
5:1.1.1.1:4::0::1234::00a:000::120/248
                              *[EVPN/170] 4w4d 13:29:25
                                    Fictitious
5:2.2.2.2:4::0::1234::a0a:1800::120/248
                              *[BGP/170] 3d 05:33:52, localpref 100, from 2.2.2.2
                                    AS path: I, validation- state: unverified
                                     to 10.10.1.20 via enp2s0
```

- `show route table <VRF>.inet.0`: Displays the route entries in the specified routing table.
- `show route table bgp.evpn.0`: Displays the route entries in the specified routing table.

A sample output with a local prefix is shown below.

```
host@pe1> show route table bgp.evpn.0 match-prefix 5:1.1.1.1:4::0::10.10.14.0::24

bgp.evpn.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
5:1.1.1.1:4::0::10.10.14.0::24/248
                   *[EVPN/170] 2w1d 05:11:43
                       Fictitious
```

A sample output with a remote prefix is shown below.

```
host@pe1> show route table bgp.evpn.0 match-prefix 5:2.2.2.2:4::0::10.10.24.0::24
bgp.evpn.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
5:2.2.2.2:4::0::10.10.24.0::24/248
                   *[BGP/170] 2w1d 05:11:48, localpref 100, from 2.2.2.2
                      AS path: I, validation-state: unverified
                    >  to 10.10.1.20 via enp2s0
```

- `show krt next-hop`: Displays the configured next hop.

vRouter CLI Commands

The following CLI commands can be executed on the vRouter CLI. To access the vRouter CLI, see "Access vRouter CLI" on page 331.

- `rt --get <prefix> --vrf <vrf-id> --family <inet4/inet6>`: Provides the route which is pointing to the specified IPv4 address.

  A sample output is shown below.

```
[host@pe1 /]# rt --get 10.10.24.0/24 --vrf 1
Match 10.10.24.0/24 in vRouter inet4 table 0/1/unicast
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/1/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
10.10.24.0/24           0          LPT       10020            30         -
```

- `vxlan --dump`: Provides information regarding the VNIs that are configured and the next hop.

A sample output is shown below.

```
[host@pe1 /]# vxlan --dump
VXLAN Table
VNID    NextHop
----------------
  10010    25
```

- `nh --get <nh-id>`: Provides the next hop details.

  A sample output is shown below.

```
[root@evpn-pe1-node /]# nh --get 30
Id:30         Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:5          Vrf:0
              Flags:Valid, Policy, Vxlan, Etree Root, l3_vxlan,
              Oif:1 Len:14 Data:52 54 00 78 c8 f2 52 54 00 ee 83 cd 08 00 Sip:1.1.1.1
Dip:2.2.2.2
              L3_Vxlan_SMac:48:5a:0d:78:78:d7 L3_Vxlan_DMac:48:5a:0d:49:fc:63
```

- `vif --list`: Provides a list of enterprises configured with the `vif`.
- `flow --l`: Displays all the active flows in the system.

  Use this command to verify the traffic flowing between CE1 and CE2 on the vRouter. A sample output is shown below.

```
[host@pe1 /]# flow -l
Flow table(size 161218560, entries 629760)

Entries: Created 11 Added 11 Deleted 20 Changed 26Processed 11 Used Overflow entries 0
(Created FlOwS/CPU: 0 0 0 0 0 0 0 0 0 0 11 0 (oflows 0)

Action: F-Forward, D=Drop N=NAT(S-SNAT, D=DNAT, PS=SPAT, Pd=DPAT, L=Link Local Port)
 Other: K(nh)=Key Nexthop, S(nh)=RPF Nexthop
 Flags: E-Evicted, Ec-Evict Candidate, N=New Flow, M-Modified Dm=Delete Marked
TCP(r=reverse): S-SYN, F=FIN, R=RST, C-HalfClose, E-Established, D=Dead
 Stats: Packets/Bytes

Index                   Source: Port/Destination: Port                      Proto(V)
-----------------------------------------------------------------------------------
95644<=>443840          10.10.24.21:30                                      1 (1)
```

```
                        10.10.14.11:0
 (Gen: 1, K(nh): 8, Action:F, Flags:, 005: -1, S(nh):30, Stats: 16/1344,
  SPort 56932, TTL 0. Sinfo 2.2.2.2)

443840<=>95644          10.10.14.11:30                                      1 (1)
                        10.10.24.21:0
 (Gen: 1, K(nh):8, Action:F, Flags:, QOS: -1, S(nh):41, Stats: 16/1344,
  SPort 53983, TTL 0, Sinfo 0.0.0.0)
```

- `vifdump <vif-number>`: Displays all the packet details for the specified `vif`.

  A sample output is shown below.

```
[host@pe1 /]# vifdump 3 -nevv
vif0/3      PCI: 0000:04:00.0 NH: 8 MTU: 9000
dropped privs to tcpdump
tcpdump: listening on mon3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
20:15:15.611827 52:54:00:2c:f6:16 > 52:54:00:ef:3c:4d, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 64, id 1764, offset 0, flags [DF], proto ICMP (1), length 84)
    10.10.14.11 > 10.10.24.21: ICMP echo request, id 16, seq 25, length 64
20:15:15.612472 52:54:00:ef:3c:4d > 52:54:00:2c:f6:16, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 62, id 14142, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.24.21 > 10.10.14.11: ICMP echo reply, id 16, seq 25, length 64
20:15:16.626773 52:54:00:2c:f6:16 > 52:54:00:ef:3c:4d, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 64, id 1863, offset 0, flags [DF], proto ICMP (1), length 84)
    10.10.14.11 > 10.10.24.21: ICMP echo request, id 16, seq 26, length 64
20:15:16.627404 52:54:00:ef:3c:4d > 52:54:00:2c:f6:16, ethertype IPv4 (0x0800), length 98:
(tos 0x0, ttl 62, id 14187, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.24.21 > 10.10.14.11: ICMP echo reply, id 16, seq 26, length 64
```

Integrated Routing and Bridging on JCNR

IN THIS SECTION

Integrated Routing and Bridging (IRB) is a networking concept that combines the functionalities of routing and bridging within a single network infrastructure. This integration allows for seamless communication between devices on different network segments or subnets.

In a router, packets are forwarded based on their destination IP addresses. Routers operate at Layer 3 (Network Layer) of the OSI model and make decisions about the best path for a packet to reach its destination. In a bridge, frames are forwarded based on MAC addresses. Bridges operate at Layer 2 (Data Link Layer) and use MAC addresses to determine the appropriate segment for a frame.

IRB combines the features of routing and bridging in a single device, typically a router. This allows the device to make forwarding decisions based on both IP addresses and MAC addresses. IRB is particularly useful when you want to enable communication between devices on different subnets in a network. It allows the router to route traffic between subnets based on IP addresses. Instead of having separate routers and bridges, IRB simplifies network design by consolidating these functions into a single device. In VLAN environments, each VLAN can be considered a separate subnet, and the router with IRB capability can route traffic between these VLANs.

Starting with Juniper Cloud-Native Router (JCNR) Release 23.4, Cloud-Native Router supports IRB, using which you can configure both routing and bridging settings in a unified manner. You can configure IRB interfaces and connect Bridge Domains (BD's) to perform routing between bridge domains.

To learn more about IRB, see *Integrated Routing and Bridging*.

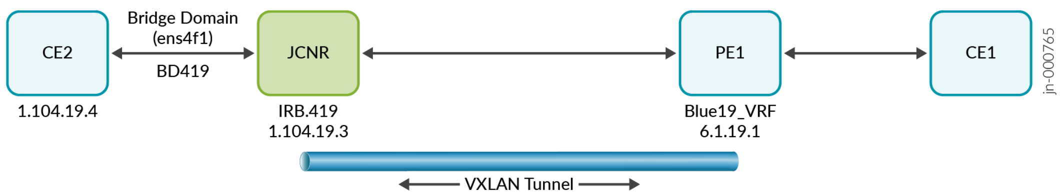
NOTE:

- Configurable MAC address on IRB is not supported in Cloud-Native Router Release 23.4
- MTU is not configurable on IRB
- BGP unnumbered is not supported on IRB interfaces

Configuring IRB

A pair of IRB interfaces are created for each BD, one for host connectivity (i.e., tap IRB interface) and another for forwarding traffic on fabric (i.e., fabric IRB interface). A single tap interface is created per L2 instance and all tap IRB interfaces that are configured in that L2 instance are created as sub-interfaces on that tap interface.

Consider the following topology shown below and configure IRB on JCNR.



Configuring an IRB interface

> **NOTE**: Use the *configlet resource* to configure the cRPD pods.

```
interfaces {
    irb {
        unit 419 {
            family inet {
                address 1.104.19.3/24;
            }
            family inet6 {
                address 2419::3/64;
            }
        }
    }
    ens4f1 {
        unit 0 {
            family bridge {
                interface-mode trunk;
                vlan-id-list [ 100 200 400 414-423 500 ];
            }
        }
    }
}
```

Attaching an IRB interface as an L3-routing interface to a Bridge

Attach an IRB interface as an L3-routing interface to a Bridge using the example below.

```
routing-instances {
    vswitch {
```

```
        instance-type virtual-switch;
        bridge-domains {
            bd419 {
                vlan-id 419;
                routing-interface irb.419;
            }
        }
        interface ens4f1;
    }
}
```

Attaching an IRB interface to VRF

An IRB interface can be a part of VRF-0 or VRF-N. The example shown below demonstrates how you can attach IRB.419 to a VRF Blue19.

```
routing-instances {
    blue19 {
        instance-type vrf;
        protocols {
            bgp {
                group ce_pe_19_v4 {
                    type external;
                    local-address 1.104.19.3;
                    peer-as 1002;
                    local-as 64512;
                    bfd-liveness-detection {
                        minimum-interval 300;
                    }
                    neighbor 1.104.19.4;
                }
                group ce_pe_19_v6 {
                    type external;
                    local-address 2419::3;
                    peer-as 1002;
                    local-as 64512;
                    bfd-liveness-detection {
                        minimum-interval 300;
                    }
                    neighbor 2419::4;
                }
```

```
            }
            evpn {
                ip-prefix-routes {
                    advertise direct-nexthop;
                    encapsulation vxlan;
                    vni 2019;
                    export vrf_route_19;
                }
            }
        }
        interface irb.419;
        interface lo0.19;
        route-distinguisher 100.100.100.1:2019;
        vrf-target target:20:2019;
    }
}
```

Troubleshooting IRB

Use the commands listed in the sections below to troubleshoot an IRB setup.

cRPD CLI Commands

The following CLI commands can be executed on the cRPD CLI. To access the cRPD CLI, see "Access cRPD CLI" on page 329.

- `show bridge mac-table vlan-id <id>`: Provides the Bridge MAC table details.

```
root@jcnr> show bridge mac-table vlan-id 419

MAC flags         (S - Static MAC, D - Dynamic MAC)
Routing Instance : default-domain:contrail:ip-fabric:default
Bridging domain VLAN id : 419
MAC                    MAC               Logical
address                flags             interface

02:22:ec:ac:6b:24      D                  irb.419
e4:5d:37:2b:2a:aa      D                  ens4f1
```

- `show bgp summary`: Provides a summary of the BGP session running on the IRB.

```
root@jcnr> show bgp summary

Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
1.104.19.4             1002        284        280       0       0     2:11:02 Establ
  blue19.inet.0: 9/9/9/0
2419::4                1002        283        280       0       0     2:10:58 Establ
  blue19.inet6.0: 9/9/9/0
```

- `show bfd session`: Provides a summary of the BFD session running on the IRB.

```
root@jcnr> show bfd session
                                                  Detect   Transmit
Address                  State     Interface      Time     Interval  Multiplier
1.104.19.4               Up        irb.419        0.900     0.300        3
2419::4                  Up        irb.419        0.900     0.300        3
```

- `ping routing-instance blue19 1.104.19.3 source 1.104.19.4 count 1 rapid`: Provides a confirmation of the network connectivity to the IRB interface from CE2.

```
root@CE2> ping routing-instance blue19 1.104.19.3 source 1.104.19.4 count 1 rapid
PING 1.104.19.3 (1.104.19.3): 56 data bytes
!
--- 1.104.19.3 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 9.041/9.041/9.041/0.000 ms
```

- `ping routing-instance blue19 6.1.19.1 source 1.104.19.4 count 1 rapid`: Provides a confirmation of the network connectivity to remote prefixes from CE2 through the IRB interface.

```
root@CE2> ping routing-instance blue19 6.1.19.1 source 1.104.19.4 count 1 rapid
PING 6.1.19.1 (6.1.19.1): 56 data bytes
!
--- 6.1.19.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 17.773/17.773/17.773/0.000 ms
```

- `traceroute routing-instance blue19 6.1.19.1 source 1.104.19.4 no-resolve`: Provides the trace routes to remote prefixes from CE2 through the IRB interface.

```
root@CE2> traceroute routing-instance blue19 6.1.19.1 source 1.104.19.4 no-resolve
traceroute to 6.1.19.1 (6.1.19.1) from 1.104.19.4, 30 hops max, 52 byte packets
1  1.104.19.3  14.341 ms  14.932 ms  14.997 ms
2  6.1.19.1  14.962 ms  9.985 ms  14.906 ms
```

vRouter CLI Commands

The following CLI commands can be executed on the vRouter CLI. To access the vRouter CLI, see "Access vRouter CLI" on page 331.

- `vif --list | grep <interface ID>`: Provides the VIF ID of the specified interface.

```
bash-5.1# vif --list | grep irb.419
vif0/26      Virtual: irb.419 NH: 73
vif0/27      Virtual: irb.419 Vlan(o/i)(,S): 419/419
```

- `vif --get 26`: Provides the VRF ID where IRB.419 is attached to.

```
bash-5.1# vif --get 26
vif0/26      Virtual: irb.419 NH: 73
             Type:Irb HWaddr:02:22:ec:ac:6b:24 IPaddr:1.104.19.3
             IP6addr:2419::3
             DDP: OFF SwLB: ON
             Vrf:2 Mcast Vrf:2 Flags:L3L2DProxyEr QOS:-1 Ref:16
             RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
             Vlan Mode: Access  Vlan Id: 419  OVlan Id: 419
             RX packets:66910  bytes:5409152 errors:0
             TX packets:71340  bytes:5718843 errors:0
             Drops:9

bash-5.1# vif --get 27
vif0/27      Virtual: irb.419 Vlan(o/i)(,S): 419/419
             Parent:vif0/9  Sub-type:  host-irb-tap
             Type:Virtual(Vlan) HWaddr:02:22:ec:ac:6b:24 IPaddr:1.104.19.3
             IP6addr:2419::3
             DDP: OFF SwLB: ON
             Vrf:2 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:1 TxXVif:26
```

```
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:71248  bytes:5711219 errors:0
              TX packets:66828  bytes:5134644 errors:0
              Drops:0
```

- `rt --get 1.104.19.4/32 --vrf 2`: Provides the data plane encapsulation for CE2's IP.

```
bash-5.1# rt --get 1.104.19.4/32 --vrf 2
Match 1.104.19.4/32 in vRouter inet4 table 0/2/unicast

Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/2/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
1.104.19.4/32           0           PT           -             113        -

bash-5.1# nh --get 113
Id:113        Type:Encap          Fmly:AF_INET/6  Rid:0  Ref_cnt:11         Vrf:2
              Flags:Valid, Policy, Etree Root,
              EncapFmly:0806 Oif:26 Len:14
              Encap Data: e4 5d 37 2b 2a aa 02 22 ec ac 6b 24
```

- `purel2cli --mac show`: Provides the MAC table in the vRouter.

```
bash-5.1# purel2cli --mac show | grep 419
02:22:ec:ac:6b:24 419        26         1
e4:5d:37:2b:2a:aa 419        3          68174
```

L3 Routing Protocols

SUMMARY

Read this topic to know about the L3 routing protocols that are supported by the Juniper Cloud Native Router, including BGP, IS-IS, and OSPF.

IN THIS SECTION

NOTE: Use the *configlet resource* to configure the cRPD pods.

BGP

BGP is an exterior gateway protocol (EGP) that is used to exchange routing information among routers in different autonomous systems (ASs). BGP routing information includes the complete route to each destination. BGP uses the routing information to maintain a database of network reachability information, which it exchanges with other BGP systems. BGP uses the network reachability information to construct a graph of AS connectivity, which enables BGP to remove routing loops and enforce policy decisions at the AS level. The cloud-native router supports BGP version 4. Here is an example to configure BGP protocol on the cloud-native router:

```
set protocols bgp group CNI type internal
set protocols bgp group CNI local-address 10.0.0.1
set protocols bgp group CNI family inet-vpn unicast
set protocols bgp group CNI family inet6-vpn unicast
set protocols bgp group CNI neighbor 10.0.1.1 peer-as 64512
set protocols bgp group CNI neighbor 10.0.1.1 local-as 64512
set routing-options route-distinguisher-id 10.0.0.1
```

You can issue the `show bgp summary` command on the cRPD shell to view the BGP summary information for all routing instances. For example:

```
user@host> show bgp summary
Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 1 Peers: 1 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.l3vpn.0
                       2          2          0          0          0          0
bgp.l3vpn-inet6.0
                       2          2          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
10.0.1.1             64512        249        211       0       0     1:32:42 Establ
```

```
bgp.l3vpn.0: 2/2/2/0
bgp.l3vpn-inet6.0: 2/2/2/0
jcnr-3.inet.0: 2/2/2/0
jcnr-3.inet6.0: 2/2/2/0
```

Refer the BGP User Guide for more information.

IS-IS

The IS-IS protocol is an interior gateway protocol (IGP) that uses link-state information to make routing decisions. IS-IS is a link-state IGP that uses the shortest-path-first (SPF) algorithm to determine routes. IS-IS evaluates the topology changes and determines whether to perform a full SPF recalculation or a partial route calculation (PRC). IS-IS uses hello packets that allow network convergence to occur quickly when network changes are detected. The cloud-native router supports IS-IS.

Here is an example to configure IS-IS protocol on the cloud-native router:

```
set security forwarding-options family iso mode packet-based
set interfaces eno3v0 unit 0 family inet address 10.100.12.1/30
set interfaces eno3v0 unit 0 family iso
set interfaces lo0 unit 0 family inet address 192.168.0.1/32
set interfaces lo0 unit 0 family iso address 49.0002.0192.0168.0001.00
set protocols isis interface eno3v0
set protocols isis interface lo0.0
```

You can issue the `show isis adjacency` and `show isis interface` commands to verify the protocol configuration. Refer the IS-IS User Guide for information.

OSPF

OSPF is an interior gateway protocol (IGP) that routes packets within a single autonomous system (AS). OSPF uses link-state information to make routing decisions, making route calculations using the shortest-path-first (SPF) algorithm (also referred to as the Dijkstra algorithm). Each router running OSPF floods link-state advertisements throughout the AS or area that contain information about that router's attached interfaces and routing metrics. Each router uses the information in these link-state advertisements to calculate the least cost path to each network and create a routing table for the protocol. The cloud-native router supports OSPF version 2 (OSPFv2) and OSPF version 3 (OSPFv3). Here is an example to configure IS-IS protocol on the cloud-native router:

```
set protocols ospf area 0.0.0.0 interface bond0
set protocols ospf area 0.0.0.0 interface lo passive
```

Once you bring up the pods, verify the OSPF configuration:

```
show ospf neighbor
Address          Interface              State           ID               Pri  Dead
192.168.123.254  bond0                  Full            123.1.1.254      128    36
```

```
user@host> show route 1.1.24.24

inet.0: 27 destinations, 29 routes (27 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.1.24.24/32       *[OSPF/10] 00:07:08, metric 2
                    >  to 192.168.123.254 via bond0
```

Refer the OSPF User Guide for more information.

MPLS Support

IN THIS SECTION

The Juniper Cloud-Native Router contains support for MPLS routing protocols. You use the JCNR-controller, or cRPD, to configure MPLS using the node annotations at the time of deployment or via the "cRPD CLI" on page 329.

The cRPD then sends the configuration to the vRouter-agent, using gRPC. The vRouter-agent then converts the configuration to network policies that it imlements in the vRouter. The cloud-native router supports the following MPLS-based routing protocols:

MPLS Support

- **L3 MPLS VPN (MPLS)**—L3 MPLS VPNs are also known as BGP/MPLS VPNs because BGP is used to distribute VPN routing information across the provider's backbone, and MPLS is used to forward VPN traffic across the backbone to remote VPN sites. The cloud-native router can particpate as a sending, receiving or transit router using the MPLS protocol. Review the L3 VPN User Guide for more information.

- **Segment Routing-MPLS (SR-MPLS)**—Review the "Segment Routing" on page 120 topic for more detail.

- **MPLS over UDP (MPLSoUDP)**—MPLSoUDP is an overlay technology that encapsulates MPLS packets within UDP packets to traverse through some networks that do not support native MPLS or SR-MPLS. The cloud-native router can participate as a sending, receiving or transit router using MPLSoUDP. Review the Configuring Next-Hop-Based MPLSoUDP Tunnels topic for a configuration example. Both IPv4 and IPv6 tunnels are supported.

- **Label Distribution Protocol (LDP)**—The Label Distribution Protocol (LDP) is a protocol for distributing labels in non-traffic-engineered applications. LDP allows routers to establish label-switched paths (LSPs) through a network by mapping network-layer routing information directly to data link layer-switched paths. The cloud-native router can participate as a sending, receiving or transit router using LDP. Review the LDP Overview topic for more information.

Bidirectional Forwarding Detection (BFD)

SUMMARY

Read this topic to know about the support for Bidirectional Forwarding Detection (BFD) in the Juniper Cloud-Native router.

IN THIS SECTION

The Bidirectional Forwarding Detection (BFD) protocol is a simple hello mechanism that detects failures in a network. A pair of routing devices exchange BFD packets. The devices send hello packets at a specified, regular interval. The device detects a neighbor failure when the routing device stops receiving a reply after a specified interval. The cloud-native router supports both single-hop and multihop BFD. Review the Understanding BFD topic for more information.

Configuration

The following configlet configures BFD on internal BGP peer session with BGP neighbor `192.168.1.3`.

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set interfaces ens1f2 unit 0 family inet address 10.10.1.26/24
    set interfaces ens1f2 unit 0 family inet6 address 2001:0db8:192:168:1::2610:10:1::26/64
    set interfaces ens1f2 unit 0 family mpls
    set interfaces lo0 unit 0 family inet address 192.168.1.26/32
    set interfaces lo0 unit 0 family inet6 address 2001:0db8:192:168:1::26/128
    set protocols ospf area 0 interface lo0.0
    set protocols ospf area 0 interface ens1f2
    set protocols ospf3 area 0 interface lo0.0
    set protocols ospf3 area 0 interface ens1f2
    set protocols mpls interface ens1f2
    set protocols mpls interface lo0.0
    set protocols bgp group BFD type internal
    set protocols bgp group BFD multihop
    set protocols bgp group BFD local-address 192.168.1.26
    set protocols bgp group BFD family inet-vpn unicast
    set protocols bgp group BFD family inet6-vpn unicast
    set protocols bgp group BFD local-as 64512
    set protocols bgp group BFD neighbor 192.168.1.3 bfd-liveness-detection minimum-interval 2500
    set protocols bgp group BFDv6 type internal
    set protocols bgp group BFDv6 family inet6-vpn unicast
    set protocols bgp group BFDv6 local-as 64512
    set protocols bgp group BFDv6 neighbor 2001:0db8:192:168:1::3 local-address
2001:0db8:192:168:1::26
```

```
    set protocols bgp group BFDv6 neighbor 2001:0db8:192:168:1::3 bfd-liveness-detection minimum-
interval 2500
    set routing-options dynamic-tunnels to_pe source-address 192.168.1.26
    set routing-options dynamic-tunnels to_pe udp
    set routing-options dynamic-tunnels to_pe destination-networks 192.168.1.3/32
    set protocols mpls ipv6-tunneling
    set protocols bfd traceoptions file bfd.log
    set protocols bfd traceoptions flag all
  crpdSelector:
    matchLabels:
        node: worker
```

Verification

You can verify the BFD configuration on the "cRPD shell" on page 329.

Verify BGP Summary

Verify the BGP summary using the `show bgp summary` command.

```
user@host> show bgp summary
Threading mode: BGP I/O
Default eBGP mode: advertise - accept, receive - accept
Groups: 2 Peers: 2 Down peers: 0
Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
bgp.l3vpn.0
                      0          0          0          0          0          0
bgp.l3vpn-inet6.0
                      0          0          0          0          0          0
Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|#Active/
Received/Accepted/Damped...
192.168.1.3             64512         13         11       0       1        4:02 Establ
  bgp.l3vpn.0: 0/0/0/0
  bgp.l3vpn-inet6.0: 0/0/0/0
2001:0db8:192:168:1::3                64512         13         12       0       1        4:49
Establ
  bgp.l3vpn-inet6.0: 0/0/0/0
```

Verify BFD Sessions

Verify BFD sessions between the iBGP peers using the `show bfd session` command.

```
user@host> show bfd session
                                                     Detect   Transmit
Address                       State     Interface      Time     Interval     Multiplier
192.168.1.3                   Up                       0.900     0.300       3
2001:0db8:192:168:1::3                  Up             0.900     0.300       3

2 sessions, 2 clients
Cumulative transmit rate 6.7 pps, cumulative receive rate 6.7 pps
```

Virtual Router Redundancy Protocol (VRRP)

SUMMARY

Read this topic to learn about the support for the Virtual Router Redundancy Protocol (VRRP) in Juniper Cloud-Native router.

The Virtual Router Redundancy Protocol (VRRP) enables hosts on a LAN to make use of redundant routing platforms on that LAN without requiring more than the static configuration of a single default route on the hosts. The VRRP routing platforms share the IP address corresponding to the default route configured on the hosts. At any time, one of the VRRP routing platforms is the primary (active) and the others are backups. If the primary routing platform fails, one of the backup routing platforms becomes the new primary routing platform, providing a virtual default routing platform and enabling traffic on the LAN to be routed without relying on a single routing platform. Using VRRP, a backup device can take over a failed default device within a few seconds. This is done with minimum VRRP traffic and without any interaction with the hosts. When Cloud-Native Router is deployed in the containerized network function (CNF) mode in cloud deployments, the VRRP unicast can be used to decide between the active and backup Cloud-Native Router nodes. Review the Understanding VRRP topic for more information.

NOTE: To enable VRRP for Cloud-Native Router on an EKS cluster, a ConfigMap must be configured. Please review *Cloud-Native Router ConfigMap for VRRP* topic for more information

Virtual Routing Instance (VRF-Lite)

SUMMARY

Read this topic to understand the implementation of virtual routing instances in JCNR.

IN THIS SECTION

Virtual routing instances allow administrators to divide the cloud-native router into multiple independent virtual routers, each with its own routing table. Splitting a device into many virtual routing instances isolates traffic traveling across the network without requiring multiple devices to segment the network. You can use virtual routing instances to isolate customer traffic on your network and to bind customer-specific instances to customer-owned interfaces. Virtual routing and forwarding (VRF) is often used in conjunction with Layer 3 subinterfaces, allowing traffic on a single physical interface to be differentiated and associated with multiple virtual routers. Each logical Layer 3 subinterface can belong to only one routing instance. Review the Virtual Router Instances topic for more information.

Configuration

You can create a virtual routing instance in Cloud-Native Router via a network attachment definition (NAD) manifest. Here is an example NAD to create a `bluenet` virtual router routing instance:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: blue
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "blue-net",
```

```
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "bluenet",
          "instanceType": "virtual-router"
        },
        "kubeConfig":"/root/.kube/config"
      }
    ]
  }'
```

Note the `instanceType` is set to `virtual-router`. Refer to "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 for more information on NAD.

Here is an example configuration for a `podblue` pod with an interface (`192.168.11.10/24`) attached to the `blue` network (output is trimmed for brevity):

```
apiVersion: v1
kind: Pod
metadata:
  name: podblue
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "blue",
          "interface":"net1",
          "cni-args": {
            "interfaceType":"veth",
            "dataplane":"dpdk",
            "mac":"aa:bb:cc:dd:ee:10",
            "ipConfig":{
              "ipv4":{
                "address":"192.168.11.10/24",
                "gateway":"192.168.11.1",
                "routes":["192.168.11.0/24"]
              },
              "ipv6":{
                "address":"abcd::192.168.11.10/112",
                "gateway":"abcd::192.168.11.1",
                "routes":["abcd::192.168.11.0/112"]
```

```
                }
              }
            }
          }
        ]
spec:
...
```

As you apply the NAD and the pod manifests using the `kubectl apply -f` *`manifest`* command, the `bluenet` routing instance and `bluenet.inet.0` routing table is created in the Cloud-Native Router controller. You can configure Cloud-Native Router to enable communication from `podblue` to pods on the remote network.

NOTE: Use the *configlet resource* to configure the cRPD pods.

Here is an example cRPD configuration:

1. Configure the local fabric interface and the BGP protocol:

   ```
   set interfaces ens2f0 unit 0 family inet address 10.10.10.11/24
   set protocols bgp group overlay type internal
   set protocols bgp group overlay local-address 10.10.10.11
   set protocols bgp group overlay local-as 64520
   set protocols bgp group overlay neighbor 10.10.10.12 peer-as 64520
   ```

   where `10.10.10.12/24` is the IP address of the BGP peer or neighbor router.

2. Export the `inet` routes using the BGP protocol:

   ```
   set policy-options policy-statement send_direct term 1 from protocol direct
   set policy-options policy-statement send_direct term 1 then accept
   set policy-options policy-statement send_direct term reject then reject
   set protocols bgp group overlay export send_direct
   ```

3. Leak the routes from the `bluenet` routing instance to the `default` routing instance:

   ```
   set groups cni routing-instances bluenet routing-options interface-routes rib-group inet
   blue_to_inet
   set routing-options rib-groups blue_to_inet import-rib bluenet.inet.0
   set routing-options rib-groups blue_to_inet import-rib inet.0
   ```

4. Leak only the BGP routes matching prefix `192.168.12.0` from `inet.0` to the `bluenet` routing instance, where `192.168.12.0/24` is the remote pod network:

```
set policy-options policy-statement inet_to_blue term from_bgp from instance master
set policy-options policy-statement inet_to_blue term from_bgp from protocol bgp
set policy-options policy-statement inet_to_blue term from_bgp from route-filter
192.168.12.0/24 orlonger
set policy-options policy-statement inet_to_blue term from_bgp then accept
set policy-options policy-statement inet_to_blue term reject then reject
set routing-options rib-groups inet_to_blue import-rib inet.0
set routing-options rib-groups inet_to_blue import-rib bluenet.inet.0
set routing-options rib-groups inet_to_blue import-policy inet_to_blue
set groups cni routing-instances bluenet routing-options instance-import inet_to_blue
```

NOTE: Cloud-Native Router supports route leaking between virtual router routing instances for routes with interface, receive, resolve and table next-hops.

RELATED DOCUMENTATION

Rib-Groups

ECMP

SUMMARY

Read this topic to know about the support for ECMP with flow stickiness in the Juniper Cloud-Native Router.

Equal-cost multipath (ECMP) is a network routing strategy that allows for traffic of the same session, or flow—that is, traffic with the same source and destination—to be transmitted across multiple paths of equal cost. It is a mechanism that allows you to load balance traffic and increase bandwidth by fully utilizing otherwise unused bandwidth on links to the same destination.

When forwarding a packet, the routing technology must decide which next-hop path to use. In making a determination, the device takes into account the packet header fields that identify a flow. When ECMP is used, next-hop paths of equal cost are identified based on routing metric calculations and hash algorithms. That is, routes of equal cost have the same preference and metric values, and the same cost to the network. The ECMP process identifies a set of routers, each of which is a legitimate equal cost next hop towards the destination. The routes that are identified are referred to as an ECMP set. Because it addresses only the next hop destination, ECMP can be used with most routing protocols.

An equal-cost multipath (ECMP) set is formed when the routing table contains multiple next-hop addresses for the same destination with equal cost. (Routes of equal cost have the same preference and metric values.) If there is an ECMP set for the active route, the Cloud-Native Router uses a consistent hash to choose *one* of the next-hop addresses from the ECMP members to forward the packet.

The cloud-native router supports ECMP for both Container Network Interface (CNI) and transit router modes. It supports flow stickiness when the number of next-hops is changed. The cloud-native router also supports ECMP next-hop for tunneled traffic.

BGP Unnumbered

SUMMARY

Read this topic to know about the support for BGP unnumbered in the cloud-native router.

Juniper Cloud-Native Router supports BGP unnumbered peering starting in Release 23.2. This feature allows BGP to auto-discover and to create peer neighbor sessions using the link-local IPv6 addresses of directly connected neighbors. Using BGP unnumbered peering, which dynamically discovers IPV6 neighbors, reduces the burden of manually configuring an IPv6 underlay. It is used in N-tier Clos architecture for point-to-point links. BGP unnumbered is supported in the default VRF (VRF-0) and virtual routing instances (virtual-router). Read the BGP Unnumbered topic for more information.

NOTE: When a BGP unnumbered IPv6 session is established between 2 provider edge routers (PEs) and IPv4 routes are being exchanged over that session, then the next hop for an IPv4 route is an IPv6 address. This feature is supported on PEs having Linux kernel version 5 and above. If the Linux kernel version is below 5, then the IPv4 routes are not added to the routing table.

Layer-3 VLAN Sub-Interfaces

IN THIS SECTION

VLAN sub-interfaces are like logical interfaces on a physical switch or router. They access only tagged packets that match the configured VLAN tag. A sub-interface has a parent interface. A parent interface can have multiple sub-interfaces, each with a VLAN ID. When you run the cloud-native router, you must associate each sub-interface with a specific VLAN. Starting in Juniper Cloud-Native Router Release 23.2, the cloud-native router supports the use of VLAN sub-interfaces in L3 mode along with the previously supported L2 mode.

Configuration Example

The VLAN sub-interfaces are configured using the Netowrk Attachment Definition (NAD) and pod YAML manifests. Please see the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 and relevant configuration examples for more information.

The Cloud-Native Router controller interface configuration viewed using the `show configuration` command is as shown below (the output is trimmed for brevity).

For L3 mode:

```
enp24s0f0 {
        unit 1 {
            vlan-id 10;
            family inet {
                address 172.168.20.3/24;
            }
        }
}
```

On the vRouter, a VLAN sub-interface configuration is as shown below:

For L3 mode:

```
vif0/9        Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/2 NH: 36 MTU: 1514
              Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
              IP6addr:fe80::d493:87ff:fe91:456c
              DDP: OFF SwLB: ON
              Vrf:1 Mcast Vrf:1 Flags:L3DProxyEr QOS:-1 Ref:4
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:0  bytes:0 errors:0
              TX packets:0  bytes:0 errors:0
              Drops:0

vif0/10       Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/5 NH: 21 MTU: 9000
              Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
              IP6addr:fe80::d493:87ff:fe91:456c
              DDP: OFF SwLB: ON
              Vrf:1 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:4 TxXVif:9
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:0  bytes:0 errors:0
              TX packets:0  bytes:0 errors:0
              Drops:0
```

Enabling Dynamic Device Personalization (DDP) on Individual Interfaces

SUMMARY

Dynamic Device Personalization (DDP) is a technology that enables programmable packet processing pipeline provided by Intel as a profile to their NICs. Cloud-Native Router supports enabling Dynamic Device Personalization (DDP) on individual interfaces.

Starting with Juniper Cloud-Native Router (JCNR) Release 23.2, Cloud-Native Router supports enabling Dynamic Device Personalization (DDP) on individual interfaces. This feature is available on Cloud-Native Router in L2, L3, and L2-L3 modes.

Dynamic Device Personalization (DDP) is a technology that enables programmable packet processing pipeline provided by Intel as a profile to their NICs. Multiple Intel NICs support this technology. The support varies based on the Intel NIC type. DDP is used in packet classification where the profiles applied to the NIC can classify multiple packet formats on the NIC enabling speeds and feeds to the Data Plane Development Kit (DPDK).

Juniper cloud native router (JCNR) provides routing and switching functionality. Cloud-Native Router supports interfaces from different NIC cards. Some of the Intel NICs support DDP and some of them don't support DDP. Therefore, in a deployment scenario, Cloud-Native Router might have one interface from one NIC that supports DDP and another interface from a different NIC that does not support DDP. Cloud-Native Router supports enabling DDP per interface to overcome such issues.

NOTE: For E810 PF, Cloud-Native Router loads the DDP package which is bundled with JCNR. However, for other NICs, ensure you load the DDP package on the NICs before starting JCNR.

A DDP configuration is available per interface. This configuration option overrides global DDP (`ddp`) configuration for that interface. If you do not configure an interface DDP, then the global configuration value serves as the value for that interface. If you do not configure the global DDP configuration, then the default value for the global configuration which is `off` takes effect.

NOTE: DDP is supported on the following NICs:

- E810 VF
- E810 PF
- X710 PF
- XXV710 PF

DDP support is not available when interfaces are defined under subnets.

You should configure DDP in the helm chart before deployment. Configuring the DDP configurations in the helm charts for both global and at interface levels is optional. If you do not configure the DDP keys, then the default value for global DDP which is `off` takes effect.

The global DDP configuration is available in the `values.yaml` file as shown below:

```
# Set ddp to enable Dynamic Device Personalization (DDP)
# Provides datapath optimization at NIC for traffic like GTPU, SCTP etc.
# Options include auto or on or off; default: off
ddp: "auto"
```

You can configure one of the following options for `ddp` at the interface level:

1. Auto—when set to auto, Cloud-Native Router checks if the NIC supports DDP or not during deployment and configures DPDK accordingly. Detecting whether a NIC supports DDP at run time makes is easier to deploy Cloud-Native Router in volumes.

2. On—option enables DDP on the interface without validating the NIC. Use this option only if you are sure that the NIC supports DDP.

3. Off—is the default option at the interface level. This option disables DDP on the interface.

For example,

```
-   eth1:
       ddp: "off" ## auto or on or off
```

NOTE: Each interface can have a different configuration for `ddp`. DDP is enabled for a bond interface only if all the slave interface NICs support DDP.

4

CHAPTER

Cloud-Native Router CNI Configuration Examples

IN THIS CHAPTER

Cloud-Native Router Use-Cases and Configuration Overview

SUMMARY

Read this chapter to review configuration examples for various Juniper Cloud-Native Router use cases when deployed in the container network interface (CNI) mode.

IN THIS SECTION

The Juniper Cloud-Native Router can be deployed as a virtual switch or a transit router, either as a pure container network function (CNF) or as a container network interface (CNI). In the CNF mode, there are no application pods running on the node and the router only performs packeting switching or forwarding through various interfaces on the system. In the CNI mode, application pods using software-based network interfaces such as veth-pairs or DPDK vhost-user based interfaces, attach to the cloud-native router. This chapter provides configuration examples for attaching different workload interface types to the cloud-native router CNI instance.

Configuration Example

The Cloud-Native Router CNI is deployed as a secondary CNI along with Multus as a primary CNI, to create different types of secondary interfaces for the application pod. Multus uses a network attachment definition (NAD) file to configure a secondary interface for the application pod. The NAD specifies how to create a secondary interface, IP address allocation, network instance and more. A pod can have one or more NADs, typically one per pod interface. The `config:` field in the NAD file defines the Cloud-Native Router CNI configuration. Here is a generic format of the NAD:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <vrf-name>
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "<vrf-name>",
    "plugins": [
      {
```

```
        "type": "jcnr",
        "args": {
           "key1":"value1",
           "key2","value2",
           ....
        },
        "ipam": {
          "type": "<ipam-type>",
          ....
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

While configuring the NAD for the Cloud-Native Router plugin type, the following keys are supported:

Table 23: Supported Keys in NAD

Key	Description
instanceName	The routing-instance name
instanceType	One of: virtual-router—for non-VPN-related applications vrf—Layer 3 VPN implementations virtual-switch—Layer 2 implementations
interfaceType	Either "veth" or "virtio"
vlanId	A valid vlan id "1-4095"
bridgeVlanId	A valid vlan id "1-4095"
vlanIdList	A list of command separated vlan-id, e.g: "1, 5, 7, 10-20"
parentInterface	Valid interface name as it should appear in the pod. Child/sub-interfaces have parentInterface as their prefix followed by "." If parentInterface is specified, sub interface must be explicitly specified.
vrfTarget	The route-target for vrf routing instance

Table 23: Supported Keys in NAD *(Continued)*

Key	Description
bridgeDomain	Bridge Domain under which pod interface should be attached in the virtual-switch instance.
type (ipam)	static—assigns same IP to all pods, to assign a unique IP per pod define a unique NAD per pod per interface host-local—unique IP address per pod interface on the same host. IP addresses are not unique across two different nodes whereabouts—unique IP address per pod across all nodes (https://github.com/k8snetworkplumbingwg/whereabouts)

Consider the example NAD for a layer 2 kernel access mode interface:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch-pod1-bd100
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch-pod1-bd100",
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
       "interfaceType": "veth",
          "bridgeDomain": "bd100",
          "bridgeVlanId": "100"
        },
        "ipam": {
          "type": "static",
          "addresses":[
            {
              "address":"99.61.0.2/16",
              "gateway":"99.61.0.1"
            },
```

```
            {
              "address":"1234::99.61.0.2/120",
              "gateway":"1234::99.61.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

The pod attaches to the router instance using the `k8s.v1.cni.cncf.io/networks` annotation. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: vswitch-pod1-bd100
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - kind-worker
  containers:
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
```

```
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
  volumes:
    - name: dpdk
      hostPath:
        path: /var/run/jcnr/containers
```

The volume mount host path exposes the UNIX domain socket of the vhost-user port to the DPDK application. The DPDK interface details are stored at `/dpdk/dpdk-interfaces.json` inside the application container for the DPDK application to consume. It is also exported into the pod as a pod annotation.

When you create a pod for use in the cloud-native router, the Kubernetes component known as **kubelet** calls the Multus CNI to set up pod networking and interfaces. Multus reads the annotations section of the **pod.yaml** file to refer the corresponding NAD. If a NAD points to `jcnr` as the CNI plug in, Multus calls the JCNR-CNI to set up the pod interface. JCNR-CNI creates the interface as specified in the NAD. JCNR-CNI then generates and pushes a configuration into cRPD.

Troubleshooting

Pods main fail to come up for various reasons:

- Image not found
- CNI failed to add interfaces
- CNI failed to push configuration into cRPD
- CNI failed to invoke vRouter REST APIs
- The NAD is invalid or undefined

The following commands will be useful to troubleshooting pod issues:

```
# Check the Pod status
kubectl get pods -A
```

```
# Check pod state and CNI logs
kubectl describe pod <pod-name>
```

```
# Check the pod logs
kubectl logs pod <pod-name>
```

```
# Check the net-attach-def
kubectl get net-attach-def <net-attach-def-name> -o yaml
```

```
# Check CNI logs
tail -f /var/log/jcnr/jcnr-cni.log
```

```
# Check the cRPD config added by CNI (on the cRPD CLI)
cli> show configuration groups cni
```

L2 Pod with Kernel Interface (Access Mode)

SUMMARY

Read this topic to learn how to add a user pod with a `kernel/veth` access-mode interface to an instance of the cloud-native router.

IN THIS SECTION

Overview

You can configure a user pod with a Layer 2 access-mode `kernel` interface and attach it to the Cloud-Native Router instance. The Juniper Cloud-Native Router must have an L2 interface configured at the time of deployment. Your high-level tasks are:

- Define and apply a network attachment definition (NAD)—The NAD file defines the required configuration for Multus to invoke the JCNR-CNI and create a network to attach the pod interface to.
- Define and apply a pod YAML file to your cloud-native router cluster—The pod YAML contains the pod specifications and an annotation to the network created by the JCNR-CNI.

NOTE: Please review the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 topic for more information on NAD and pod YAML files.

Configuration Example

1. Here is an example NAD to create a Layer 2 `kernel/veth` access-mode interface with static IPAM:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch-pod1-bd100
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch-pod1-bd100",
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
       "interfaceType": "veth",
          "bridgeDomain": "bd100",
          "bridgeVlanId": "100"
        },
        "ipam": {
```

```
          "type": "static",
          "addresses":[
            {
              "address":"99.61.0.2/16",
              "gateway":"99.61.0.1"
            },
            {
              "address":"1234::99.61.0.2/120",
              "gateway":"1234::99.61.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

The NAD defines a bridge domain `bd100` under which a `veth` type pod interface should be attached in the `virtual-switch` instance.

It also defines a static IP address to be assigned to the pod interface.

2. Apply the NAD manifest to create the network.

```
kubectl apply -f nad-access_mode.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vswitch-pod1-bd100 created
```

3. Verify the NAD is created.

```
[root@jcnr-01]# kubectl get net-attach-def
NAME                 AGE
vswitch-pod1-bd100   59s
```

4. Here is an example yaml to create a pod attached to the `vswitch-pod1-bd100` network:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: vswitch-pod1-bd100
```

```
spec:
  containers:
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
  volumes:
    - name: dpdk
      hostPath:
        path: /var/run/jcnr/containers
```

The pod attaches to the router instance using the `k8s.v1.cni.cncf.io/networks` annotation

.

5. Apply the pod manifest.

```
[root@jcnr-01]# kubectl apply -f pod_access_mode.yaml
pod/pod1 created
```

6. Verify the pod is running.

```
[root@jcnr-01 ~]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          2m38s
```

7. Describe the pod to verify a secondary interface is created and attached to the `vswitch-pod1-bd100` network. (The output is trimmed for brevity).

```
[root@jcnr-01 ~]# kubectl describe pod pod1
Name:         pod1
```

```
Namespace:    default
Priority:     0
Node:         jcnr-01/10.100.20.25
Start Time:   Mon, 26 Jun 2023 09:36:57 -0400
Labels:       <none>
Annotations:  cni.projectcalico.org/containerID:
5b92668a6d7580e587de951d660c99969ce98bc239502afab6f9d191653f1e9b
              cni.projectcalico.org/podIP: 10.233.91.79/32
              cni.projectcalico.org/podIPs: 10.233.91.79/32
              k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "k8s-pod-network",
                    "ips": [
                        "10.233.91.79"
                    ],
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/vswitch-pod1-bd100",
                    "interface": "net1",
                    "ips": [
                        "99.61.0.2",
                        "1234::633d:2"
                    ],
                    "mac": "02:00:00:5D:74:76",
                    "dns": {}
                }]
...
```

8. Verify the vRouter has the corresponding interface created. "Access the vRouter CLI" on page 331 and issue the `vif --list` command.

```
vif0/2      Ethernet: jvknet1-7c557fe MTU: 9160
            Type:Virtual HWaddr:02:00:00:66:01:56
            DDP: OFF SwLB: ON
            Vrf:0 Flags:L2Vof QOS:-1 Ref:8
            RX port   packets:20 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Vlan Mode: Access  Vlan Id: 100  OVlan Id: 100
            RX packets:7  bytes:518 errors:13
            TX packets:31  bytes:2438 errors:0
```

```
            Drops:14
            TX port    packets:31 errors:0
```

Note that the interface type is `Virtual` and the Vlan mode is set to `access` with the Vlan ID set to `100`. The VRF is always 0 for L2 interfaces.

L2 Pod with virtio Interface (Trunk Mode)

SUMMARY

Read this topic to learn how to add a user pod with a `virtio` trunk-mode interface to an instance of the cloud-native router.

IN THIS SECTION

Overview

You can configure a user pod with a Layer 2 trunk-mode virtio interface and attach it to the Cloud-Native Router instance. The Juniper Cloud-Native Router must have an L2 interface configured at the time of deployment. Your high-level tasks are:

- Define and apply a network attachment definition (NAD)—The NAD file defines the required configuration for Multus to invoke the JCNR-CNI and create a network to attach the pod interface to.
- Define and apply a pod YAML file to your cloud-native router cluster—The pod YAML contains the pod specifications and an annotation to the network created by the JCNR-CNI.

NOTE: Please review the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 topic for more information on NAD and pod YAML files.

NOTE: When deploying a virtio application pod in privileged mode on Microsoft Azure Cloud Platform, it should be compiled with DPDK version greater than 23.11. While invoking the DPDK application, the fabric interfaces used by Cloud-Native Router should be blocked out, for example:

```
./dpdk_pod_23.11  -b vmbus:000d3a9d-4df3-000d-3a9d-4df3000d3a9d
```

Configuration Example

1. Here is an example NAD to create a Layer 2 trunk-mode virtio interface with static IPAM:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch",
    "type": "jcnr",
    "args": {
      "instanceName": "vswitch",
      "instanceType": "virtual-switch",
      "vlanIdList":"201, 202, 203"
    },
    "ipam": {
         "type": "static",
         "capabilities":{"ips":true},
         "addresses":[
           {
             "address":"10.2.1.1/24",
             "gateway":"10.2.1.253"
           },
           {
             "address":"2001::10.2.1.1/120",
             "gateway":"2001::10.2.1.253"
           }
         ]
    },
```

```
    "kubeConfig":"/etc/kubernetes/kubelet.conf"
  }'
```

The NAD defines the VLAN IDs for the `virtual-switch` instance to which the pod's trunk interface will be attached.

2. Apply the NAD manifest to create the network.

```
kubectl apply -f nad_trunk_mode.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vswitch created
```

3. Verify the NAD is created.

```
[root@jcnr-01]# kubectl get net-attach-def
NAME                 AGE
vswitch              57s
```

4. Here is an example yaml to create a pod attached to the `vswitch` network:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: vswitch
spec:
  containers:
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
```

```
          subPathExpr: $(KUBERNETES_POD_UID)
    volumes:
      - name: dpdk
        hostPath:
          path: /var/run/jcnr/containers
```

The pod attaches to the router instance using the `k8s.v1.cni.cncf.io/networks` annotation.

5. Apply the pod manifest.

```
[root@jcnr-01]# kubectl apply -f pod_trunk_mode.yaml
pod/pod1 created
```

6. Verify the pod is running.

```
[root@jcnr-01 ~]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          38s
```

7. Describe the pod to verify a secondary interface is created and attached to the `vswitch` network. (The output is trimmed for brevity).

```
[root@jcnr-01 ~]# kubectl describe pod pod1
Name:         pod1
Namespace:    default
Priority:     0
Node:         jcnr-01/10.100.20.25
Start Time:   Mon, 26 Jun 2023 09:53:31 -0400
Labels:       <none>
Annotations:  cni.projectcalico.org/containerID:
ac6f0a26ebfe68adf3b020d0def96f09e6b2b5c6303f55c0dde277b1ce7f9d9f
              cni.projectcalico.org/podIP: 10.233.91.81/32
              cni.projectcalico.org/podIPs: 10.233.91.81/32
              jcnr.juniper.net/dpdk-interfaces:
                [
                    {
                        "name": "net1",
                        "vhost-adaptor-path": "/dpdk/vhost-net1.sock",
                        "vhost-adaptor-mode": "client",
                        "ipv4-address": "10.2.1.1/24",
```

```
                    "ipv6-address": "2001::a02:101/120",
                    "mac-address": "02:00:00:5B:C7:9F"
                }
            ]
          k8s.v1.cni.cncf.io/network-status:
            [{
                "name": "k8s-pod-network",
                "ips": [
                    "10.233.91.81"
                ],
                "default": true,
                "dns": {}
            },{
                "name": "default/vswitch",
                "interface": "net1",
                "ips": [
                    "10.2.1.1",
                    "2001::a02:101"
                ],
                "mac": "02:00:00:5B:C7:9F",
                "dns": {}
            }]
...
```

8. Verify the vRouter has the corresponding interface created. "Access the vRouter CLI" on page 331 and issue the `vif --list` command.

```
vif0/2      PMD: vhostnet1-57f38cc0-6555-4bc2-ac MTU: 9160
            Type:Virtual HWaddr:02:00:00:dc:c9:27
            DDP: OFF SwLB: ON
            Vrf:0 Flags:L2 QOS:-1 Ref:11
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Vlan Mode: Trunk  Vlan: 201-203
            RX packets:0  bytes:0 errors:0
            TX packets:4  bytes:256 errors:0
            Drops:0
            TX port   packets:0 errors:4
```

Note that the interface type is `Virtual` and the Vlan mode is set to `trunk` with the Vlan ID set to `201-203`. The VRF is always 0 for L2 interfaces.

L2 Pod with VLAN Sub-Interface

SUMMARY

Read this topic to learn how to add a user pod with a Layer 2 VLAN sub-interface to an instance of the cloud-native router.

IN THIS SECTION

Overview

You can configure a user pod with a Layer 2 VLAN sub-interface and attach it to the Cloud-Native Router instance. The Juniper Cloud-Native Router must have an L2 interface configured at the time of deployment. The cRPD must be configured with the valid VLAN configuration for the fabric interface. For example:

```
set interfaces eth1 unit 100 vlan-id 100
```

NOTE: Note that the unit number and the VLAN ID must match.

Your high-level tasks are:

- Define and apply a network attachment definition (NAD)—The NAD file defines the required configuration for Multus to invoke the JCNR-CNI and create a network to attach the pod interface to.
- Define and apply a pod YAML file to your cloud-native router cluster—The pod YAML contains the pod specifications and an annotation to the network created by the JCNR-CNI

NOTE: Please review the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 topic for more information on NAD and pod YAML files.

Configuration Example

1. Here is an example NAD to create a Layer 2 VLAN sub-interface:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch-bd201-sub
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch-bd201-sub",
    "capabilities":{"ips":true},
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
          "bridgeDomain": "bd201",
          "bridgeVlanId": "201",
          "parentInterface": "net1",
          "interface": "net1.201"
        },
        "ipam": {
          "type": "static",
          "capabilities":{"ips":true},
          "addresses":[
            {
              "address":"10.3.0.1/24",
              "gateway":"10.3.0.254"
            },
            {
              "address":"2001:db8:3003::10.3.0.1/120",
              "gateway":"2001:db8:3003::10.3.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
```

```
    ]
  }'
```

The NAD defines a bridge domain `bd201` and a sub-interface `net1.201` with a parent interface `net1`. The pod will be attached in the `virtual-switch` instance.. It also defines a static IP address to be assigned to the pod interface.

2. Apply the NAD manifest to create the network.

```
kubectl apply -f nad_l2_vlan_subinterface.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vswitch-bd201-sub created
```

3. Verify the NAD is created.

```
[root@jcnr-01]# kubectl get net-attach-def
NAME                 AGE
vswitch-bd201-sub    43s
```

4. Here is an example yaml to create a pod attached to the `vswitch-bd201-sub` network:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: "vswitch-bd201-sub"
spec:
  containers:
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      resources:
        requests:
          memory: 2Gi
        limits:
          hugepages-1Gi: 2Gi
      env:
        - name: KUBERNETES_POD_UID
```

```
            valueFrom:
              fieldRef:
                 fieldPath: metadata.uid
        volumeMounts:
          - name: dpdk
            mountPath: /dpdk
            subPathExpr: $(KUBERNETES_POD_UID)
          - mountPath: /dev/hugepages
            name: hugepage
    volumes:
      - name: dpdk
        hostPath:
          path: /var/run/jcnr/containers
      - name: hugepage
        emptyDir:
          medium: HugePages
```

The pod attaches to the router instance using the `k8s.v1.cni.cncf.io/networks` annotation.

5. Apply the pod manifest.

```
[root@jcnr-01]# kubectl apply -f pod_access_mode.yaml
pod/pod1 created
```

6. Verify the pod is running.

```
[root@jcnr-01 ~]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          40s
```

7. Describe the pod to verify a secondary interface is created and attached to the `vswitch-bd201-sub` network. (The output is trimmed for brevity).

```
[root@jcnr-01 ~]# kubectl describe pod pod1
Name:         pod1
Namespace:    default
Priority:     0
Node:         jcnr-01/10.100.20.25
Start Time:   Mon, 26 Jun 2023 09:53:31 -0400
Labels:       <none>
```

```
Annotations:  cni.projectcalico.org/containerID:
58642dd26f85769e14d302153357e84e6900398532d1b82b50a845ac1ede051a
              cni.projectcalico.org/podIP:
              cni.projectcalico.org/podIPs:
              jcnr.juniper.net/dpdk-interfaces:
                [
                    {
                        "name": "net1",
                        "vhost-adaptor-path": "/dpdk/vhost-net1.sock",
                        "vhost-adaptor-mode": "client",
                        "ipv4-address": "10.3.0.1/24",
                        "ipv6-address": "2001:db8:3003::a03:1/120",
                        "mac-address": "02:00:00:84:DC:42",
                        "vlan-id": "201"
                    }
                ]
              k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "k8s-pod-network",
                    "ips": [
                        "10.233.91.97"
                    ],
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/vswitch-bd201-sub",
                    "interface": "net1",
                    "ips": [
                        "10.3.0.1",
                        "2001:db8:3003::a03:1"
                    ],
                    "mac": "02:00:00:84:DC:42",
                    "dns": {}
                }]
...
```

8. Verify the vRouter has the corresponding interface created. "Access the vRouter CLI" on page 331 and issue the `vif --list` command.

```
vif0/2      PMD: vhostnet1-d5eee4ec-dd7c-4e MTU: 9160
            Type:Virtual HWaddr:02:00:00:84:dc:42
            DDP: OFF SwLB: ON
```

```
              Vrf:65535 Flags:L2 QOS:-1 Ref:14
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:0  bytes:0 errors:0
              TX packets:0  bytes:0 errors:0
              Drops:0
              TX port   packets:0 errors:293

vif0/3        Virtual: vhostnet1-d5eee4ec-dd7c-4e.201 Vlan(o/i)(,S): 201/201 Parent:vif0/2 MTU:
1514
              Type:Virtual(Vlan) HWaddr:02:00:00:84:dc:42
              DDP: OFF SwLB: ON
              Vrf:0 Flags:L2 QOS:-1 Ref:1
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:0  bytes:0 errors:0
              TX packets:208  bytes:17071 errors:0
              Drops:0
```

Note that the interface type is `Virtual` and the Vlan ID set to `201`. The parent interface is `vif0/2`. The VRF is always 0 for L2 sub-interfaces.

L3 Pod with VPN Interface

SUMMARY

Read this topic to learn how to add a user pod with a `virtio` and `kernel` interfaces attached to an L3 VPN instance on the cloud-native router.

IN THIS SECTION

Overview

You can configure a user pod with a `virtio` and `kernel` interfaces to an L3 VPN instance on the cloud-native router. The Juniper Cloud-Native Router must have an L3 interface configured at the time of deployment. Your high-level tasks are:

- Define and apply a network attachment definition (NAD)—The NAD file defines the required configuration for Multus to invoke the JCNR-CNI and create a network to attach the pod interface to.
- Define and apply a pod YAML file to your cloud-native router cluster—The pod YAML contains the pod specifications and an annotation to the network created by the JCNR-CNI.

NOTE: Please review the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 topic for more information on NAD and pod YAML files.

Configuration Example

1. Here is an example NAD to create a `virtio` interface attached to an L3 VPN instance:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vrf100
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vrf100",
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vrf100",
          "instanceType": "vrf",
          "vrfTarget":"100:1"
        },
        "ipam": {
          "type": "static",
          "addresses":[
            {
              "address":"99.61.0.2/16",
              "gateway":"99.61.0.1"
            },
            {
              "address":"1234::99.61.0.2/120",
```

```
                "gateway":"1234::99.61.0.1"
              }
            ]
          },
          "kubeConfig":"/etc/kubernetes/kubelet.conf"
        }
      ]
    }'
```

The NAD defines a virtual routing and forwarding (VRF) instance `vrf100` to which the pod's virtio interface will be attached. You must use the `vrf` instance type for Layer 3 VPN implementations. The NAD also defines a static IP address to be assigned to the pod interface.

2. Apply the NAD manifest to create the network.

```
kubectl apply -f nad_virtio_L3vpn.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vrf100 created
```

3. Here is an example NAD to create a `kernel` interface attached to an L3VPN instance:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vrf200
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vrf200",
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vrf200",
          "instanceType": "vrf",
       "interfaceType": "veth",
          "vrfTarget":"200:1"
        },
        "ipam": {
          "type": "static",
          "addresses":[
            {
```

```
              "address":"99.62.0.2/16",
              "gateway":"99.62.0.1"
            },
            {
              "address":"1234::99.62.0.2/120",
              "gateway":"1234::99.62.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

The NAD defines a virtual routing and forwarding (VRF) instance `vrf200` with a `veth` interface type to which the pod's kernel interface will be attached.

It also defines a static IP address to be assigned to the pod interface.

4. Apply the NAD manifest to create the network.

```
kubectl apply -f nad_kernel_L3vpn.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vrf200 created
```

5. Verify the NADs are created.

```
[root@jcnr-01]# kubectl get net-attach-def
NAME                 AGE
vrf100               8m40s
vrf200               55s
```

6. Here is an example yaml to create a pod attached to the `vrf100` and `vrf200` networks:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: vrf100, vrf200
spec:
  containers:
```

```
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
  volumes:
    - name: dpdk
      hostPath:
        path: /var/run/jcnr/containers
```

The pod attaches to the router instance using the `k8s.v1.cni.cncf.io/networks` annotation.

7. Apply the pod manifest.

```
[root@jcnr-01]# kubectl apply -f pod_access_mode.yaml
pod/pod1 created
```

8. Verify the pod is running.

```
[root@jcnr-01 ~]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          2m38s
```

9. Describe the pod to verify two secondary interface are created and attached to the `vrf100` and `vrf200` networks. (The output is trimmed for brevity).

```
[root@jcnr-01 ~]# kubectl describe pod pod1
Name:          pod1
Namespace:     default
Priority:      0
Node:          jcnr-01/10.100.20.25
```

```
Start Time:   Mon, 26 Jun 2023 09:53:31 -0400
Labels:       <none>
Annotations:  cni.projectcalico.org/containerID:
6705c204abca5aeaa0241c1791ea911d57bd972336d969ac5d6a482c96348d95
              cni.projectcalico.org/podIP: 10.233.91.100/32
              cni.projectcalico.org/podIPs: 10.233.91.100/32
              jcnr.juniper.net/dpdk-interfaces:
                [
                    {
                        "name": "net1",
                        "vhost-adaptor-path": "/dpdk/vhost-net1.sock",
                        "vhost-adaptor-mode": "client",
                        "ipv4-address": "99.61.0.2/16",
                        "ipv6-address": "1234::633d:2/120",
                        "mac-address": "02:00:00:A9:B3:23"
                    }
                ]
              k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "k8s-pod-network",
                    "ips": [
                        "10.233.91.100"
                    ],
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/vrf100",
                    "interface": "net1",
                    "ips": [
                        "99.61.0.2",
                        "1234::633d:2"
                    ],
                    "mac": "02:00:00:A9:B3:23",
                    "dns": {}
                },{
                    "name": "default/vrf200",
                    "interface": "net2",
                    "ips": [
                        "99.62.0.2",
                        "1234::633e:2"
                    ],
                    "mac": "02:00:00:E0:AC:59",
                    "dns": {}
```

```
                }]
...
```

10. Verify the vRouter has the corresponding interface created. "Access the vRouter CLI" on page 331 and issue the `vif --list` command.

```
vif0/5      PMD: vhostnet1-2464783d-1ddd-4bf5-b7 NH: 16 MTU: 9160
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:99.61.0.2
            IP6addr:1234::633d:2
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:PL3DProxyEr QOS:-1 Ref:14
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/6      Ethernet: jvknet2-2464783 NH: 19 MTU: 9160
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:99.62.0.2
            IP6addr:1234::633e:2
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:PL3DVofProxyEr QOS:-1 Ref:11
            RX port   packets:28 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:28  bytes:13612 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:28
```

Note that the interface type is `Virtual` and the type of interface is L3. You can see the IP addresses assigned to the interfaces for the corresponding valid VRF numbers.

L3 Pod with VLAN Sub-Interface

SUMMARY

IN THIS SECTION

Read this topic to learn how to add a user pod with a Layer 3 VLAN sub-interface to an instance of the cloud-native router.

Overview

You can configure a user pod with a Layer 3 VLAN sub-interface and attach it to the Cloud-Native Router instance. The Juniper Cloud-Native Router must have an L3 interface configured at the time of deployment. The cRPD must be configured with the valid VLAN configuration for the fabric interface. For example:

```
set interfaces ens1f1v1 unit 201 vlan-id 201
set interfaces ens1f1v1 unit 201 family inet address 192.168.123.1/24
set interfaces ens1f1v1 unit 201 family inet6 address abcd:192:168:123::1/64
set routing-instance blue interface ens1f1v1.201
```

Your high-level tasks are:

- Define and apply a network attachment definition (NAD)—The NAD file defines the required configuration for Multus to invoke the JCNR-CNI and create a network to attach the pod interface to.
- Define and apply a pod YAML file to your cloud-native router cluster—The pod YAML contains the pod specifications and an annotation to the network created by the JCNR-CNI

NOTE: Please review the "Cloud-Native Router Use-Cases and Configuration Overview " on page 224 topic for more information on NAD and pod YAML files.

Configuration Example

1. Here are example NADs to create a Layer 3 VLAN sub-interface:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vrf201
```

```
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vrf201",
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vrf201",
          "instanceType": "virtual-router",
          "parentInterface":"net1",
          "vlanId": "201"
        },
        "ipam": {
          "type": "static",
          "addresses":[
            {
              "address":"99.61.0.2/16",
              "gateway":"99.61.0.1"
            },
            {
              "address":"1234::99.61.0.2/120",
              "gateway":"1234::99.61.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

The NAD defines virtual-router instances `vrf201` with the parent interface `net1` and VLAN ID `201`. A `virtual-router` instance type is similar to a VPN routing and forwarding instance type, but used for non-VPN-related applications. There are no virtual routing and forwarding (VRF) import, VRF export, VRF target, or route distinguisher requirements for this instance type. The pod VLAN sub-interface is attached to `vrf201` instance. The NAD also defines static IP addresses to be assigned to the pod interface.

2. Apply the NAD manifests to create the networks.

```
kubectl apply -f nad_l3_vlan_subinterface_201.yaml
networkattachmentdefinition.k8s.cni.cncf.io/vrf201 created
```

3. Verify the NADs are created.

```
kubectl get net-attach-def
NAME      AGE
vrf201    30s
```

4. Here is an example yaml to create a pod attached to the `vrf201` and `vrf202` networks:

```
apiVersion: v1
kind: Pod
metadata:
  name:   pod1
  annotations:
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "vrf201",
          "interface":"net1.201"
        }
      ]
spec:
  containers:
    - name: pod1
      image: ubuntu:latest
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
  volumes:
    - name: dpdk
      hostPath:
        path: /var/run/jcnr/containers
```

The pod attaches to the router instances using the `k8s.v1.cni.cncf.io/networks` annotation.

5. Apply the pod manifest.

```
[root@jcnr-01]# kubectl apply -f pod_l3_subinterface.yaml
pod/pod1 created
```

6. Verify the pod is running.

```
[root@jcnr-01 ~]# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          38s
```

7. Describe the pod to verify a secondary interface is created and attached to the `vrf201` network. (The output is trimmed for brevity).

```
[root@jcnr-01 ~]# kubectl describe pod pod1
Name:         pod1
Namespace:    default
Priority:     0
Node:         jcnr-01/10.100.20.25
Start Time:   Mon, 26 Jun 2023 09:53:31 -0400
Labels:       <none>
Annotations:  cni.projectcalico.org/containerID:
90de252886b3e0a97526ac175544078fb03debf05650946d759e2de0d5179c17
              cni.projectcalico.org/podIP: 10.233.91.126/32
              cni.projectcalico.org/podIPs: 10.233.91.126/32
              jcnr.juniper.net/dpdk-interfaces:
                [
                    {
                        "name": "net1.201",
                        "vhost-adaptor-path": "/dpdk/vhost-net1.sock",
                        "vhost-adaptor-mode": "client",
                        "ipv4-address": "99.61.0.2/16",
                        "ipv6-address": "1234::633d:2/120",
                        "mac-address": "02:00:00:8C:97:A2",
                        "vlan-id": "201"
                    }
                ]
              k8s.v1.cni.cncf.io/network-status:
```

```
                [{
                    "name": "k8s-pod-network",
                    "ips": [
                        "10.233.91.126"
                    ],
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/vrf201",
                    "interface": "net1.201",
                    "ips": [
                        "99.61.0.2",
                        "1234::633d:2"
                    ],
                    "mac": "02:00:00:8C:97:A2",
                    "dns": {}
                }]
...
```

8. Verify the vRouter has the corresponding interface created. "Access the vRouter CLI" on page 315 and issue the `vif --list` command.

```
vif0/11      PCI: 0000:b3:11.1 (Speed 10000, Duplex 1) NH: 16 MTU: 9014          ---> fabric
interface
             Type:Physical HWaddr:b2:56:78:5c:af:fa IPaddr:0.0.0.0
             DDP: OFF SwLB: ON
             Vrf:0 Mcast Vrf:0 Flags:L3L2Vof QOS:0 Ref:42
             RX port   packets:10988509 errors:0
             RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
             Fabric Interface: 0000:b3:11.1  Status: UP  Driver: net_iavf
             RX packets:10988509  bytes:5582067106 errors:0
             TX packets:10988484  bytes:5581953776 errors:0
             Drops:0
             TX port   packets:10988484 errors:0

vif0/17      PMD: ens1f1v1 NH: 44 MTU: 9000                                     ---> tap
interface
             Type:Host HWaddr:b2:56:78:5c:af:fa IPaddr:0.0.0.0
             DDP: OFF SwLB: ON
             Vrf:0 Mcast Vrf:0 Flags:L3L2 QOS:0 Ref:41 TxXVif:11
             RX device packets:2201  bytes:935980 errors:0
             RX queue  packets:2201 errors:0
```

```
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:2201  bytes:935980 errors:0
            TX packets:493  bytes:161906 errors:0
            Drops:0
            TX queue  packets:493 errors:0
            TX device packets:493  bytes:161906 errors:0

vif0/48     Virtual: ens1f1v1.201 Vlan(o/i)(,S): 201/201 NH: 161 MTU: 1514
            Parent:vif0/11  Sub-type:  physical-tap                            ---> L3 sub-
interface, parent is a physical interface
            Type:Virtual(Vlan) HWaddr:b2:56:78:5c:af:fa IPaddr:192.168.123.1
            IP6addr:abcd:192:168:123::1
            DDP: OFF SwLB: ON
            Vrf:201 Mcast Vrf:201 Flags:L3DProxyEr QOS:-1 Ref:4
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:18  bytes:1836 errors:0
            Drops:0

vif0/49     Virtual: ens1f1v1.201 Vlan(o/i)(,S): 201/201 NH: 156 MTU: 9000
            Parent:vif0/17  Sub-type:  Host-tap                                ---> L3 sub-
interface, parent is a tap interface
            Type:Virtual(Vlan) HWaddr:b2:56:78:5c:af:fa IPaddr:192.168.123.1
            IP6addr:abcd:192:168:123::1
            DDP: OFF SwLB: ON
            Vrf:201 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:4 TxXVif:48
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:18  bytes:1908 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/50     PMD: vhostnet1-9403fd77-648a-47 NH: 177 MTU: 9160                     ---> pod
interface
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:65535 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:20
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/51     Virtual: vhostnet1-9403fd77-648a-47.201 Vlan(o/i)(,S): 201/201 NH: 17 MTU: 1514
            Parent:vif0/50                                                     ---->L3 pod
```

```
sub-interface, parent is the pod interface
            Type:Virtual(Vlan) HWaddr:00:00:5e:00:01:00 IPaddr:99.62.0.2
            IP6addr:1234::633e:2
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:PL3DProxyEr QOS:-1 Ref:4
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0
```

You can see the IP addresses assigned to the sub-interfaces for the corresponding valid VRF numbers.

5

CHAPTER

Monitoring and Logging

IN THIS CHAPTER

Using Cloud-Native Router Controller CLI (cRPD)

SUMMARY

This topic contains instructions to access the Cloud-Native Router controller (cRPD) CLI and run operational commands.

IN THIS SECTION

Accessing the Cloud-Native Router Controller (cRPD) CLI

You can access the command-line interface (CLI) of the cloud-native router controller by accessing the shell of the running cRPD container. Refer to "Access cRPD CLI" on page 329 to learn how to access the cRPD shell.

At this point, you have connected to the shell of the cRPD. Just as with other Junos-based devices, you access the operational mode of the cloud-native router the same way as if you were connected to the console of a physical Junos OS device.

```
root@jcnr-01:/# cli
root@jcnr-cni>
```

NOTE: The `ping` command is not supported in the cRPD shell (CLI mode).

Example Show Commands

Here are some example show commands you can execute:

```
show interfaces terse
Interface@link   Oper State     Addresses
__crpd-brd1      UNKNOWN        fe80::acbf:beff:fe8a:e046/64
```

```
cali1b684d67bd4@if3 UP             fe80::ecee:eeff:feee:eeee/64
cali34cf41e29bb@if3 UP             fe80::ecee:eeff:feee:eeee/64
docker0          DOWN          172.17.0.1/16
eno1             UP            10.102.70.146/24 fe80::a94:efff:fe79:dcae/64
eno2             UP
eno3             UP            10.1.1.1/24 fe80::a94:efff:fe79:dcac/64
eno3v1           UP
eno4             DOWN
enp0s20f0u1u6    UNKNOWN
ens2f0           DOWN
ens2f1           DOWN
erspan0@NONE     DOWN
eth0             UNKNOWN       169.254.143.126/32 fe80::b4db:eeff:fe78:9f43/64
gre0@NONE        UNKNOWN
gretap0@NONE     DOWN
ip6tnl0@NONE     UNKNOWN       fe80::74b6:2cff:fea7:d850/64
irb              DOWN
kube-ipvs0       DOWN          10.233.0.1/32 10.233.0.3/32 10.233.35.229/32
lo               UNKNOWN       127.0.0.1/8 ::1/128
lsi              UNKNOWN       fe80::cc59:6dff:fe9c:4db3/64
nodelocaldns     DOWN          169.254.25.10/32
sit0@NONE
UNKNOWN        ::169.254.143.126/96 ::10.233.91.64/96 ::172.17.0.1/96 ::10.102.70.146/96 ::10.1.1
.1/96 ::127.0.0.1/96
tunl0@NONE       UNKNOWN
vxlan.calico     UNKNOWN       10.233.91.64/32 fe80::64c6:34ff:fecd:3522/64
```

```
show configuration routing-instances
vswitch {
    instance-type virtual-switch;
    bridge-domains {
        bd100 {
            vlan-id 100;
        }
        bd200 {
            vlan-id 200;
        }
        bd300 {
            vlan-id 300;
        }
        bd700 {
```

```
            vlan-id 700;
            interface enp59s0f1v0;
        }
        bd701 {
            vlan-id 701;
        }
        bd702 {
            vlan-id 702;
        }
        bd703 {
            vlan-id 703;
        }
        bd704 {
            vlan-id 704;
        }
        bd705 {
            vlan-id 705;
        }
    }
    interface bond0;
}
```

```
show bridge ?
Possible completions:
mac-table       Show media access control table
statistics      Show bridge statistics information
```

```
show bridge mac-table ?
Possible completions:
  <[Enter]>            Execute this command
  count                Number of MAC address
  mac-address          MAC address in the format XX:XX:XX:XX:XX:XX
  vlan-id              Display MAC address learned on a specified VLAN or 'all-vlan'
  |                    Pipe through a command
```

```
show bridge mac-table
Routing Instance : default-domain:default-project:ip-fabric:__default__
Bridging domain VLAN id : 3002
MAC                  MAC                Logical
```

```
address              flags              interface

00:00:5E:00:53:01      D                 bond0
```

```
show bridge statistics ?
Possible completions:
  <[Enter]>            Execute this command
  vlan-id              Display statistics for a particular vlan (1..4094)
  |                    Pipe through a command
```

```
show bridge statistics
Bridge domain vlan-id: 100
   Local interface:  bond0
      Broadcast packets Tx  : 0           Rx  : 0
      Multicast packets Tx  : 0           Rx  : 0
      Unicast packets Tx    : 0           Rx  : 0
      Broadcast bytes Tx    : 0           Rx  : 0
      Multicast bytes Tx    : 0           Rx  : 0
      Unicast bytes Tx      : 0           Rx  : 0
      Flooded packets       : 0
      Flooded bytes         : 0
   Local interface: ens1f0v1
      Broadcast packets Tx  : 0           Rx  : 0
      Multicast packets Tx  : 0           Rx  : 0
      Unicast packets Tx    : 0           Rx  : 0
      Broadcast bytes Tx    : 0           Rx  : 0
      Multicast bytes Tx    : 0           Rx  : 0
      Unicast bytes Tx      : 0           Rx  : 0
      Flooded packets       : 0
      Flooded bytes         : 0
   Local interface: ens1f3v1
      Broadcast packets Tx  : 0           Rx  : 0
      Multicast packets Tx  : 0           Rx  : 0
      Unicast packets Tx    : 0           Rx  : 0
      Broadcast bytes Tx    : 0           Rx  : 0
      Multicast bytes Tx    : 0           Rx  : 0
```

```
      Unicast bytes Tx      : 0           Rx  : 0
      Flooded packets       : 0
```

```
show firewall filter filter1
Filter : filter1    vlan-id : 3001
 Term                   Packet
  t1                    0
```

```
show configuration firewall:firewall
family {
    bridge {
        filter filter1 {
            term t1 {
                from {
                    destination-mac-address 10:30:30:30:30:31;
                    source-mac-address 10:30:30:30:30:30;
                    ether-type oam;
                }
                then {
                    discard;
                }
            }
        }
    }
}
```

```
show route 172.68.20.2/32 table nad1.inet
nad1.inet.0: 11 destinations, 15 routes (11 active, 0 holddown, 0 hidden)
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both

172.68.20.2/32     @[BGP/170] 00:00:23, localpref 100, from 1.1.1.220
                      AS path: I, validation-state: unverified
                    >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.1.220), Push 48
                    [BGP/170] 00:13:18, localpref 100, from 1.1.24.24
                      AS path: I, validation-state: unverified
                    >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.24.24), Push 16
                   #[Multipath/255] 00:00:23, metric2 2
```

```
                      via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.1.220), Push 48
                   >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.24.24), Push 16
```

```
show interfaces routing enp216s0f0
Interface        State Addresses
enp216s0f0       Up    MPLS  enabled
                       ISO   enabled
                       INET  192.168.123.3
                       INET6 2001:192:168:123::3
                       INET6 fe80::42a6:b7ff:fe2c:a448
```

```
show dynamic-tunnels database
*- Signal Tunnels #- PFE-down
Table: inet.3
Destination-network: 1.1.1.220/32
Destination-network: 1.1.24.24/32
Tunnel to: 1.1.24.24/32
  Reference count: 4
  Next-hop type: UDP (forwarding-nexthop)
    Source address: 1.1.1.35
    Next hop: v6 mapped, tunnel-composite, 0x557917afc91c, nhid 0
      VPN Label: Push 16, Reference count: 2
      Ingress Route: [OSPF] 1.1.24.24/32, via metric 2
      Traffic Statistics: Packets 0, Bytes 0
      State: Up
  Aggregate Traffic Statistics:
```

Example Clear Commands

Here are some example clear commands:

```
clear bridge mac-table ?
Possible completions:
  <[Enter]>            Execute this command
  mac-address          Clear specific MAC address
```

```
  vlan-id              Clear mac-table for a specified vlan-id (1..4094)
  |                    Pipe through a command
```

```
clear bridge statistics ?
Possible completions:
  <[Enter]>            Execute this command
  vlan-id              Clear L2 interface statistics for a specified vlan-id (1..4094)
  |                    Pipe through a command
```

Telemetry Capabilities

IN THIS SECTION

Read this topic to learn about the telemetry data available from Juniper Cloud-Native Router.

Cloud-Native Router Telemetry

Juniper Cloud-Native Router comes with telemetry capabilities that enable you to see performance metrics and telemetry data. Telemetry data is derived separately from the vRouter, cRPD and cSRX (if using "IPSec security using service chaining" on page 167). The container **telemetry-exporter** provides you this visibility via either a Prometheus-based API or gRPC Network Management Interface (gNMI). Telemetry exporter is enabled for both vRouter and cRPD by default. You can use the following snippet in `values.yaml` to configure telemetry:

```
#telemetry:
  #  disable: false
  #  metricsPort: 8072
```

```
#  logLevel: info        #Possible options: warn, warning, info, debug, trace, or verbose
#  gnmi:
#    enable: true
#vrouter:
#  telemetry:
#    metricsPort: 8070
#    logLevel: info      #Possible options: warn, warning, info, debug, trace, or verbose
#    gnmi:
#      enable: true
```

Telemetry export is disabled for cSRX by default. You can enable it in the cSRX helm chart during installation:

```
telemetry:
   enable: true
```

You must also provide the cSRX root password in base64 format to the service-chain-instance secret created before the helm installation. Please review the *Apply the cSRX License and Configure cSRX* for more details.

Telemetry via Prometheus-based API

Prometheus is an open-source systems monitoring and alerting toolkit. You can use Prometheus to retrieve telemetry data from the cloud-native router host servers and view that data in the HTTP format. A sample of Prometheus configuration looks like this:

```
- job_name: "prometheus-JCNR-1a2b3c"

# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.

static_configs:
- targets: ["<host-server-IP>:8070"]
```

For vRouter, the telemetry exporter periodically queries the Introspect on the vRouter-agent for statistics and reports metrics information in response to the Prometheus scrape requests. You can directly view the telemetry data by using the following URL: **http://*host server IP address*:8070**.

The cRPD telemetry exporter periodically queries the the cRPD pod over NETCONF for statistics and reports metrics information in response to the Prometheus scrape requests. You can directly view the telemetry data by using the following URL: **http://*host server IP address*:8072**.

The cSRX telemetry exporter periodically queries the the cSRX pod over NETCONF for statistics and reports metrics information in response to the Prometheus scrape requests. You can directly view the telemetry data by using the following URL: **http://*host server IP address*:8073**.

NOTE: If the ports `8072` or `8073` is unavailable you can choose an alternate port to collect telemetry data in the respective helm charts.

The tables below shows the sample telemetry outputs for vRouter and cRPD when using Prometheus-based API.

NOTE: We've grouped the output shown in the following table. The cloud-native router does not group or sort the output on live systems.

Table 24: Sample vRouter Telemetry Data (Prometheus-based API)

Group	Sample Output

Memory usage per vRouter	`# TYPE virtual_router_system_memory_cached_bytes gauge`
	`# HELP virtual_router_system_memory_cached_bytes Virtual router system memory cached`
	`virtual_router_system_memory_cached_bytes{vrouter_name="jcnr.example.com"} 2635970448`
	
	`# TYPE virtual_router_system_memory_buffers gauge`
	`# HELP virtual_router_system_memory_buffers Virtual router system memory buffer`
	`virtual_router_system_memory_buffers{vrouter_name="jcnr.example.com"} 32689`
	
	`# TYPE virtual_router_system_memory_bytes gauge`
	`# HELP virtual_router_system_memory_bytes Virtual router total system memory`
	`virtual_router_system_memory_bytes{vrouter_name="jcnr.example.com"} 2635970448`
	
	`# TYPE virtual_router_system_memory_free_bytes gauge`
	`# HELP virtual_router_system_memory_free_bytes Virtual router system memory free`
	`virtual_router_system_memory_free_bytes{vrouter_name="jcnr.example.com"} 2635969296`
	
	`# TYPE virtual_router_system_memory_used_bytes gauge`
	`# HELP virtual_router_system_memory_used_bytes Virtual router system memory used`
	`virtual_router_system_memory_used_bytes{vrouter_name="jcnr.example.com"} 32689`
	
	`# TYPE virtual_router_virtual_memory_kilobytes gauge`
	`# HELP virtual_router_virtual_memory_kilobytes Virtual router virtual memory`
	`virtual_router_virtual_memory_kilobytes{vrouter_name="jcnr.example.com"} 0`
	
	`# TYPE virtual_router_resident_memory_kilobytes gauge`
	`# HELP virtual_router_resident_memory_kilobytes Virtual router resident memory`
	`virtual_router_resident_memory_kilobytes{vrouter_name="jcnr.example.com"} 32689`
	
	`# TYPE virtual_router_peak_virtual_memory_bytes gauge`
	`# HELP virtual_router_peak_virtual_memory_bytes Virtual router peak virtual memory`
	`virtual_router_peak_virtual_memory_bytes{vrouter_name="jcnr.example.com"} 2894328001`

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

Packet count per interface	`# TYPE virtual_router_phys_if_input_packets_total counter`
	`# HELP virtual_router_phys_if_input_packets_total Total packets received by physical interface`
	`virtual_router_phys_if_input_packets_total{vrouter_name="jcnr.example.com",interface_name="bond0"} 1483`
	
	`# TYPE virtual_router_phys_if_output_packets_total counter`
	`# HELP virtual_router_phys_if_output_packets_total Total packets sent by physical interface`
	`virtual_router_phys_if_output_packets_total{vrouter_name="jcnr.example.com",interface_name="bond0"} 32969`
	`virtual_router_phys_if_output_packets_total{vrouter_name="jcnr.example.com",interface_name="bond0"} 1585402623`
	`virtual_router_phys_if_output_packets_total{interface_name="bond0",vrouter_name="jcnr.example.com"} 1585403344`
	
	`# TYPE virtual_router_phys_if_input_bytes_total counter`
	`# HELP virtual_router_phys_if_input_bytes_total Total bytes received by physical interface`
	`virtual_router_phys_if_input_bytes_total{interface_name="bond0",vrouter_name="jcnr.example.com"} 125558`
	`virtual_router_phys_if_input_bytes_total{vrouter_name="jcnr.example.com",interface_name="bond0"} 228300499320`
	`virtual_router_phys_if_input_packets_total{interface_name="bond0",vrouter_name="jcnr.example.com"} 1585421179`
	
	`# TYPE virtual_router_phys_if_output_bytes_total counter`
	`# HELP virtual_router_phys_if_output_bytes_total Total bytes sent by physical interface`
	`virtual_router_phys_if_output_bytes_total{vrouter_name="jcnr.example.com",interface_name="bond0"} 4597076`
	`virtual_router_phys_if_output_bytes_total{interface_name="bond0",vrouter_name="jcnr.example.com"} 228297889634`
	
	`# TYPE virtual_router_phys_if_input_errors_total counter`
	`# HELP virtual_router_phys_if_input_errors_total Total input errors on the physical interface (vRouter software errors)`
	`virtual_router_phys_if_input_errors_total{vrouter_name="node1", interface_name="enp3s0"} 10`
	
	`# TYPE virtual_router_phys_if_output_errors_total counter`
	`# HELP virtual_router_phys_if_output_errors_total Total output errors on the physical`

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	interface (vRouter software errors) virtual_router_phys_if_output_errors_total{vrouter_name="node1", interface_name="enp3s0"} 0 # TYPE virtual_router_phys_if_input_port_errors_total counter # HELP virtual_router_phys_if_input_port_errors_total Total output errors on the physical interface (DPDK driver errors) virtual_router_phys_if_input_port_errors_total{vrouter_name="node1", interface_name="enp3s0"} 22 # TYPE virtual_router_phys_if_output_port_errors_total counter # HELP virtual_router_phys_if_output_port_errors_total Total input errors on the physical interface (DPDK driver errors) virtual_router_phys_if_output_port_errors_total{vrouter_name="node1", interface_name="enp3s0"} 3 # TYPE virtual_router_phys_if_input_device_errors_total counter # HELP virtual_router_phys_if_input_device_errors_total Total input errors on the physical interface (Physical device rx/tx queue errors) virtual_router_phys_if_input_device_errors_total{vrouter_name="node1", interface_name="enp3s0"} 52 # TYPE virtual_router_phys_if_output_device_errors_total counter # HELP virtual_router_phys_if_output_device_errors_total Total input errors on the physical interface (Physical device rx/tx queue errors) virtual_router_phys_if_output_device_errors_total{vrouter_name="node1", interface_name="enp3s0"} 1 # TYPE virtual_router_phys_if_discards_total counter # HELP virtual_router_phys_if_discards_total Total input discarded packets on the physical interface virtual_router_phys_if_discards_total{vrouter_name="node1", interface_name="enp3s0"} 2 # TYPE virtual_router_phys_if_drops_total counter # HELP virtual_router_phys_if_drops_total Total dropped packets on the physical interface virtual_router_phys_if_drops_total{vrouter_name="node1", interface_name="enp3s0"} 0

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

CPU usage per vRouter	`# TYPE virtual_router_cpu_1min_load_avg gauge`
	`# HELP virtual_router_cpu_1min_load_avg Virtual router CPU 1 minute load average`
	`virtual_router_cpu_1min_load_avg{vrouter_name="jcnr.example.com"} 0.11625`
	`# TYPE virtual_router_cpu_5min_load_avg gauge`
	`# HELP virtual_router_cpu_5min_load_avg Virtual router CPU 5 minute load average`
	`virtual_router_cpu_5min_load_avg{vrouter_name="jcnr.example.com"} 0.109687`
	`# TYPE virtual_router_cpu_15min_load_avg gauge`
	`# HELP virtual_router_cpu_15min_load_avg Virtual router CPU 15 minute load average`
	`virtual_router_cpu_15min_load_avg{vrouter_name="jcnr.example.com"} 0.110156`

Drop packet count per vRouter	`# TYPE virtual_router_dropped_packets_total counter`
	`# HELP virtual_router_dropped_packets_total Total packets dropped`
	`virtual_router_dropped_packets_total{vrouter_name="jcnr.example.com"} 35850`

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

Packet count per interface per VLAN	`# TYPE virtual_router_interface_vlan_multicast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_multicast_input_packets_total Total number of multicast packets received on interface VLAN`
	`virtual_router_interface_vlan_multicast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_broadcast_output_packets_total Total number of broadcast packets sent on interface VLAN`
	`virtual_router_interface_vlan_broadcast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_broadcast_input_packets_total Total number of broadcast packets received on interface VLAN`
	`virtual_router_interface_vlan_broadcast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_multicast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_multicast_output_packets_total Total number of multicast packets sent on interface VLAN`
	`virtual_router_interface_vlan_multicast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_unicast_input_packets_total Total number of unicast packets received on interface VLAN`
	`virtual_router_interface_vlan_unicast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_flooded_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_flooded_output_bytes_total Total number of output bytes flooded to interface VLAN`
	`virtual_router_interface_vlan_flooded_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_multicast_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_multicast_output_bytes_total Total number of multicast bytes sent on interface VLAN`
	`virtual_router_interface_vlan_multicast_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_unicast_output_packets_total Total number of unicast packets sent on interface VLAN`
	`virtual_router_interface_vlan_unicast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_input_bytes_total counter`

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

<table>
<tr><th>Group</th><th>Sample Output</th></tr>
<tr><td></td><td>
<pre>
HELP virtual_router_interface_vlan_broadcast_input_bytes_total Total number of
broadcast bytes received on interface VLAN
virtual_router_interface_vlan_broadcast_input_bytes_total{interface_id="1",vlan_id="100
"} 0
TYPE virtual_router_interface_vlan_multicast_input_bytes_total counter
HELP virtual_router_interface_vlan_multicast_input_bytes_total Total number of
multicast bytes received on interface VLAN
virtual_router_interface_vlan_multicast_input_bytes_total{vlan_id="100",interface_id="1
"} 0
TYPE virtual_router_interface_vlan_unicast_input_bytes_total counter
HELP virtual_router_interface_vlan_unicast_input_bytes_total Total number of unicast
bytes received on interface VLAN
virtual_router_interface_vlan_unicast_input_bytes_total{interface_id="1",vlan_id="100"}
 0
TYPE virtual_router_interface_vlan_flooded_output_packets_total counter
HELP virtual_router_interface_vlan_flooded_output_packets_total Total number of
output packets flooded to interface VLAN
virtual_router_interface_vlan_flooded_output_packets_total{interface_id="1",vlan_id="10
0"} 0
TYPE virtual_router_interface_vlan_broadcast_output_bytes_total counter
HELP virtual_router_interface_vlan_broadcast_output_bytes_total Total number of
broadcast bytes sent on interface VLAN
virtual_router_interface_vlan_broadcast_output_bytes_total{interface_id="1",vlan_id="10
0"} 0
TYPE virtual_router_interface_vlan_unicast_output_bytes_total counter
HELP virtual_router_interface_vlan_unicast_output_bytes_total Total number of
unicast bytes sent on interface VLAN
virtual_router_interface_vlan_unicast_output_bytes_total{interface_id="1",vlan_id="100"
} 0
...
</pre>
</td></tr>
</table>

Table 24: Sample vRouter Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

L3/L4 Access List (Firewall Filter) Counters	`# TYPE virtual_router_acl_counter_packets_total counter`
	`# HELP virtual_router_acl_counter_packets_total Total packets of named counters defined in firewall configuration`
	`virtual_router_acl_counter_packets_total{family="inet", filter="icmpFilter", counter="c1"} 8`
	
	`# TYPE virtual_router_acl_counter_bytes_total counter`
	`# HELP virtual_router_acl_counter_bytes_total Total value in bytes of named counters defined in firewall configuration`
	`virtual_router_acl_counter_bytes_total{family="inet", filter="icmpFilter", counter="c1"} 11808`

Table 25: Sample cRPD Telemetry Data (Prometheus-based API)

Group	Sample Output
BGP summary	# TYPE crpd_bgp_rib_table_received_prefixes_total counter # HELP crpd_bgp_rib_table_received_prefixes_total Total number of BGP RIB table prefixes received crpd_bgp_rib_table_received_prefixes_total{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_external_prefixes gauge # HELP crpd_bgp_rib_table_external_prefixes Number of BGP RIB table external prefixes crpd_bgp_rib_table_external_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_active_external_prefixes gauge # HELP crpd_bgp_rib_table_active_external_prefixes Number of BGP RIB table active external prefixes crpd_bgp_rib_table_active_external_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_suppressed_internal_prefixes gauge # HELP crpd_bgp_rib_table_suppressed_internal_prefixes Number of BGP RIB table internal prefixes currently inactive, because of damping or other reasons crpd_bgp_rib_table_suppressed_internal_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_prefixes gauge # HELP crpd_bgp_rib_table_prefixes Number of BGP RIB table prefixes crpd_bgp_rib_table_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_active_prefixes gauge # HELP crpd_bgp_rib_table_active_prefixes Number of BGP RIB table active prefixes crpd_bgp_rib_table_active_prefixes{table="bgp.l3vpn.0",node="example.juniper.net"} 0 # TYPE crpd_bgp_rib_table_history_prefixes gauge # HELP crpd_bgp_rib_table_history_prefixes Number of BGP RIB table withdrawn prefixes stored locally to keep track of damping history crpd_bgp_rib_table_history_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_suppressed_external_prefixes gauge # HELP crpd_bgp_rib_table_suppressed_external_prefixes Number of BGP RIB table external prefixes currently inactive, because of damping or other reasons crpd_bgp_rib_table_suppressed_external_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_active_internal_prefixes gauge # HELP crpd_bgp_rib_table_active_internal_prefixes Number of BGP RIB table active internal prefixes crpd_bgp_rib_table_active_internal_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_pending_prefixes gauge # HELP crpd_bgp_rib_table_pending_prefixes Number of BGP RIB table prefixes in process by BGP import policy

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	crpd_bgp_rib_table_pending_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_accepted_prefixes_total counter # HELP crpd_bgp_rib_table_accepted_prefixes_total Total number of BGP RIB table prefixes accepted crpd_bgp_rib_table_accepted_prefixes_total{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_damped_prefixes gauge # HELP crpd_bgp_rib_table_damped_prefixes Number of BGP RIB table prefixes with a figure of merit greater than zero, but still active because the value has not reached the threshold at which suppression occurs crpd_bgp_rib_table_damped_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_accepted_external_prefixes_total counter # HELP crpd_bgp_rib_table_accepted_external_prefixes_total Total number of BGP RIB table external prefixes accepted crpd_bgp_rib_table_accepted_external_prefixes_total{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_internal_prefixes gauge # HELP crpd_bgp_rib_table_internal_prefixes Number of BGP RIB table internal prefixes crpd_bgp_rib_table_internal_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_suppressed_prefixes gauge # HELP crpd_bgp_rib_table_suppressed_prefixes Number of BGP RIB table prefixes currently inactive, because of damping or other reasons crpd_bgp_rib_table_suppressed_prefixes{node="example.juniper.net",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_rib_table_accepted_internal_prefixes_total counter # HELP crpd_bgp_rib_table_accepted_internal_prefixes_total Total number of BGP RIB table internal prefixes accepted crpd_bgp_rib_table_accepted_internal_prefixes_total{node="example.juniper.net",table="bgp.l3vpn.0"} 0 crpd_bgp_rib_table_external_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_active_external_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_suppressed_internal_prefixes{table="bgp.l3vpn-inet6.0",node="example.juniper.net"} 0 crpd_bgp_rib_table_received_prefixes_total{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_active_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_history_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_suppressed_external_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	crpd_bgp_rib_table_active_internal_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_pending_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_damped_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_accepted_external_prefixes_total{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_internal_prefixes{table="bgp.l3vpn-inet6.0",node="example.juniper.net"} 0 crpd_bgp_rib_table_accepted_prefixes_total{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_accepted_internal_prefixes_total{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_suppressed_prefixes{node="example.juniper.net",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_rib_table_received_prefixes_total{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_external_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_active_external_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_suppressed_internal_prefixes{table="bgp.evpn.0",node="example.juniper.net"} 0 crpd_bgp_rib_table_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_active_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_history_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_suppressed_external_prefixes{table="bgp.evpn.0",node="example.juniper.net"} 0 crpd_bgp_rib_table_active_internal_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_pending_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_accepted_prefixes_total{table="bgp.evpn.0",node="example.juniper.net"} 0 crpd_bgp_rib_table_damped_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_accepted_external_prefixes_total{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_internal_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_suppressed_prefixes{node="example.juniper.net",table="bgp.evpn.0"} 0 crpd_bgp_rib_table_accepted_internal_prefixes_total{node="example.juniper.net",table="bgp.evpn.0"} 0 # TYPE crpd_bgp_peer_input_messages_total counter

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	# HELP crpd_bgp_peer_input_messages_total Total number of messages received from BGP peer crpd_bgp_peer_input_messages_total{peer_address="11.11.11.11",peer_as="64512",node="example.juniper.net"} 5 # TYPE crpd_bgp_peer_output_messages_total counter # HELP crpd_bgp_peer_output_messages_total Total number of messages sent to BGP peer crpd_bgp_peer_output_messages_total{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512"} 4 # TYPE crpd_bgp_peer_route_queue_count gauge # HELP crpd_bgp_peer_route_queue_count Current number of messages that are queued to be sent to BGP peer crpd_bgp_peer_route_queue_count{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512"} 0 # TYPE crpd_bgp_peer_state gauge # HELP crpd_bgp_peer_state BGP peer state (1=Established, 2=Idle, 3=Connect, 4=Active, 5=OpenSent, 6=OpenConfirm) crpd_bgp_peer_state{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512"} 1 # TYPE crpd_bgp_peer_flaps_total counter # HELP crpd_bgp_peer_flaps_total Total number of times the BGP peer session has gone down and then come back up crpd_bgp_peer_flaps_total{peer_address="11.11.11.11",peer_as="64512",node="example.juniper.net"} 1 # TYPE crpd_bgp_peer_rib_table_accepted_prefixes_total counter # HELP crpd_bgp_peer_rib_table_accepted_prefixes_total Total number of BGP RIB table active prefixes accepted from BGP peer crpd_bgp_peer_rib_table_accepted_prefixes_total{peer_as="64512",table="bgp.l3vpn.0",node="example.juniper.net",peer_address="11.11.11.11"} 0 # TYPE crpd_bgp_peer_rib_table_suppressed_prefixes gauge # HELP crpd_bgp_peer_rib_table_suppressed_prefixes Number of BGP RIB table prefixes received from BGP peer currently inactive, because of damping or other reasons crpd_bgp_peer_rib_table_suppressed_prefixes{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.l3vpn.0"} 0 # TYPE crpd_bgp_peer_rib_table_active_prefixes gauge # HELP crpd_bgp_peer_rib_table_active_prefixes Number of BGP RIB table active prefixes received from BGP peer crpd_bgp_peer_rib_table_active_prefixes{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.l3vpn.0"} 0 crpd_bgp_peer_rib_table_active_prefixes{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_peer_rib_table_accepted_prefixes_total{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_peer_rib_table_suppressed_prefixes{node="example.juniper.net",peer_address="1

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	1.11.11.11",peer_as="64512",table="bgp.l3vpn-inet6.0"} 0 crpd_bgp_peer_rib_table_active_prefixes{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.evpn.0"} 0 crpd_bgp_peer_rib_table_accepted_prefixes_total{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.evpn.0"} 0 crpd_bgp_peer_rib_table_suppressed_prefixes{node="example.juniper.net",peer_address="11.11.11.11",peer_as="64512",table="bgp.evpn.0"} 0

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
Route table summary	# TYPE crpd_route_table_destinations gauge # HELP crpd_route_table_destinations Number of destinations for which there are routes in the routing table crpd_route_table_destinations{node="example.juniper.net",table="inet.0"} 13 # TYPE crpd_route_table_routes gauge # HELP crpd_route_table_routes Number of routes in the routing table crpd_route_table_routes{node="example.juniper.net",table="inet.0"} 15 # TYPE crpd_route_table_active_routes gauge # HELP crpd_route_table_active_routes Number of active routes in the routing table crpd_route_table_active_routes{node="example.juniper.net",table="inet.0"} 13 # TYPE crpd_route_table_holddown_routes gauge # HELP crpd_route_table_holddown_routes Number of routes in the routing table that are in the hold-down state before being declared inactive crpd_route_table_holddown_routes{node="example.juniper.net",table="inet.0"} 0 # TYPE crpd_route_table_hidden_routes gauge # HELP crpd_route_table_hidden_routes Number of routes in the routing table that are not used, because of routing policy crpd_route_table_hidden_routes{node="example.juniper.net",table="inet.0"} 0 crpd_route_table_routes{node="example.juniper.net",table="inet.3"} 1 crpd_route_table_active_routes{node="example.juniper.net",table="inet.3"} 1 crpd_route_table_holddown_routes{node="example.juniper.net",table="inet.3"} 0 crpd_route_table_hidden_routes{node="example.juniper.net",table="inet.3"} 0 crpd_route_table_destinations{node="example.juniper.net",table="inet.3"} 1 crpd_route_table_holddown_routes{node="example.juniper.net",table="mpls.0"} 0 crpd_route_table_hidden_routes{node="example.juniper.net",table="mpls.0"} 0 crpd_route_table_destinations{node="example.juniper.net",table="mpls.0"} 4 crpd_route_table_routes{node="example.juniper.net",table="mpls.0"} 4 crpd_route_table_active_routes{node="example.juniper.net",table="mpls.0"} 4 crpd_route_table_active_routes{node="example.juniper.net",table="inet6.0"} 34 crpd_route_table_holddown_routes{node="example.juniper.net",table="inet6.0"} 0 crpd_route_table_hidden_routes{node="example.juniper.net",table="inet6.0"} 0 crpd_route_table_destinations{node="example.juniper.net",table="inet6.0"} 34 crpd_route_table_routes{table="inet6.0",node="example.juniper.net"} 38 crpd_route_table_destinations{node="example.juniper.net",table="inet6.3"} 1 crpd_route_table_routes{node="example.juniper.net",table="inet6.3"} 1 crpd_route_table_active_routes{node="example.juniper.net",table="inet6.3"} 1 crpd_route_table_holddown_routes{node="example.juniper.net",table="inet6.3"} 0 crpd_route_table_hidden_routes{node="example.juniper.net",table="inet6.3"} 0 # TYPE crpd_route_table_protocol_routes gauge # HELP crpd_route_table_protocol_routes Number of routes in the routing table learned from the protocol

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

	`crpd_route_table_protocol_routes{protocol="Direct",node="example.juniper.net",table="inet.0"} 6`
	`# TYPE crpd_route_table_protocol_active_routes gauge`
	`# HELP crpd_route_table_protocol_active_routes Number of active routes in the routing table learned from the protocol`
	`crpd_route_table_protocol_active_routes{protocol="Direct",node="example.juniper.net",table="inet.0"} 6`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet.0",protocol="Local"} 3`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet.0",protocol="Local"} 5`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet.0",protocol="OSPF"} 4`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet.0",protocol="OSPF"} 4`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet.3",protocol="Tunnel"} 1`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet.3",protocol="Tunnel"} 1`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="mpls.0",protocol="MPLS"} 4`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="mpls.0",protocol="MPLS"} 4`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet6.0",protocol="Direct"} 8`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet6.0",protocol="Direct"} 4`
	`crpd_route_table_protocol_routes{table="inet6.0",protocol="Local",node="example.juniper.net"} 29`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet6.0",protocol="Local"} 29`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet6.0",protocol="INET6"} 1`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet6.0",protocol="INET6"} 1`
	`crpd_route_table_protocol_routes{node="example.juniper.net",table="inet6.3",protocol="Tunnel"} 1`
	`crpd_route_table_protocol_active_routes{node="example.juniper.net",table="inet6.3",protocol="Tunnel"} 1`

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

OSPF summary	`# TYPE crpd_ospf_packets_sent_total counter`
	`# HELP crpd_ospf_packets_sent_total Total number of OSPF packets sent`
	`crpd_ospf_packets_sent_total{node="example.juniper.net",packet_type="Hello"} 26`
	`# TYPE crpd_ospf_packets_received_total counter`
	`# HELP crpd_ospf_packets_received_total Total number of OSPF packets received`
	`crpd_ospf_packets_received_total{node="example.juniper.net",packet_type="Hello"} 4`
	`crpd_ospf_packets_sent_total{node="example.juniper.net",packet_type="DbD"} 3`
	`crpd_ospf_packets_received_total{node="example.juniper.net",packet_type="DbD"} 4`
	`crpd_ospf_packets_sent_total{node="example.juniper.net",packet_type="LSReq"} 1`
	`crpd_ospf_packets_received_total{node="example.juniper.net",packet_type="LSReq"} 1`
	`crpd_ospf_packets_sent_total{node="example.juniper.net",packet_type="LSUpdate"} 2`
	`crpd_ospf_packets_received_total{node="example.juniper.net",packet_type="LSUpdate"} 3`
	`crpd_ospf_packets_sent_total{node="example.juniper.net",packet_type="LSAck"} 3`
	`crpd_ospf_packets_received_total{node="example.juniper.net",packet_type="LSAck"} 2`
	`# TYPE crpd_ospf_dbd_packets_retransmitted_total counter`
	`# HELP crpd_ospf_dbd_packets_retransmitted_total Total number of OSPF database descriptor packets retransmitted`
	`crpd_ospf_dbd_packets_retransmitted_total{node="example.juniper.net"} 1`
	`# TYPE crpd_ospf_lsa_packets_retransmitted_total counter`
	`# HELP crpd_ospf_lsa_packets_retransmitted_total Total number of OSPF link-state advertisement packets retransmitted`
	`crpd_ospf_lsa_packets_retransmitted_total{node="example.juniper.net"} 0`
	`# TYPE crpd_ospf_lsa_packets_flooded_total counter`
	`# HELP crpd_ospf_lsa_packets_flooded_total Total number of OSPF link-state advertisement packets flooded`
	`crpd_ospf_lsa_packets_flooded_total{node="example.juniper.net"} 1`
	`# TYPE crpd_ospf_flood_queue_depth gauge`
	`# HELP crpd_ospf_flood_queue_depth Number of entries in the extended queue`
	`crpd_ospf_flood_queue_depth{node="example.juniper.net"} 0`
	`# TYPE crpd_ospf_error_total counter`
	`# HELP crpd_ospf_error_total Total number of OSPF receive errors`
	`crpd_ospf_error_total{error_type="no-error",node="example.juniper.net"} 0`
	`# TYPE crpd_ospf_neighbor_state gauge`
	`# HELP crpd_ospf_neighbor_state OSPF neighbor state (0=Down, 1=Full, 2=Attempt, 3=Exchange, 4=ExStart, 5=Init, 6=Loading, 7=2Way)`
	`crpd_ospf_neighbor_state{neighbor_address="113.113.113.3",interface_name="enp6s0",node ="example.juniper.net"} 1`

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
MPLS statistics	# TYPE crpd_mpls_ingress_lsp_sessions_down gauge # HELP crpd_mpls_ingress_lsp_sessions_down Number of MPLS ingress LSP sessions crpd_mpls_ingress_lsp_sessions_down{node="example.juniper.net"} 0 # TYPE crpd_mpls_ingress_lsp_sessions gauge # HELP crpd_mpls_ingress_lsp_sessions Number of MPLS ingress LSP sessions crpd_mpls_ingress_lsp_sessions{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_make_before_breaks_total counter # HELP crpd_mpls_lsp_make_before_breaks_total Total number of LSP make before break procedures performed crpd_mpls_lsp_make_before_breaks_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_bandwidth_increases_total counter # HELP crpd_mpls_lsp_bandwidth_increases_total Total number of LSP bandwidth increases performed crpd_mpls_lsp_bandwidth_increases_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_bandwidth_decreases_total counter # HELP crpd_mpls_lsp_bandwidth_decreases_total Total number of bandwidth decreases performed crpd_mpls_lsp_bandwidth_decreases_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_update_cspf_failures_total counter # HELP crpd_mpls_lsp_update_cspf_failures_total Total number of in-place LSP auto-bandwidth resizing failures at the CSPF path computation stage crpd_mpls_lsp_update_cspf_failures_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_update_signaling_errors_total counter # HELP crpd_mpls_lsp_update_signaling_errors_total Total number of in-place LSP auto-bandwidth resizing failures when RSVP signaling error is received crpd_mpls_lsp_update_signaling_errors_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_lsp_update_signaling_timeouts_total counter # HELP crpd_mpls_lsp_update_signaling_timeouts_total Total number of in-place LSP auto-bandwidth resizing failures when RSVP signaling takes too long to complete crpd_mpls_lsp_update_signaling_timeouts_total{node="example.juniper.net"} 0 # TYPE crpd_mpls_label_space_total_labels gauge # HELP crpd_mpls_label_space_total_labels The total label space available crpd_mpls_label_space_total_labels{label_space="LSI",node="example.juniper.net"} 999984 # TYPE crpd_mpls_label_space_free_labels gauge # HELP crpd_mpls_label_space_free_labels The number of freely available labels crpd_mpls_label_space_free_labels{node="example.juniper.net",label_space="LSI"} 999984 crpd_mpls_label_space_total_labels{node="example.juniper.net",label_space="Block"} 999984 crpd_mpls_label_space_free_labels{node="example.juniper.net",label_space="Block"} 999984

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

	`crpd_mpls_label_space_total_labels{node="example.juniper.net",label_space="Dynamic"} 999984`
	`crpd_mpls_label_space_free_labels{node="example.juniper.net",label_space="Dynamic"} 999984`
	`crpd_mpls_label_space_total_labels{node="example.juniper.net",label_space="Static"} 48576`
	`crpd_mpls_label_space_free_labels{node="example.juniper.net",label_space="Static"} 48576`

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

<table>
<tr><th>Group</th><th>Sample Output</th></tr>
<tr><td>LLDP Statistics</td><td>Global Statistics:

<code># TYPE crpd_lldp_remote_db_table_inserts_total counter
HELP crpd_lldp_remote_db_table_inserts_total Number of insertions made in the remote database table
crpd_lldp_remote_db_table_inserts_total 192
TYPE crpd_lldp_remote_db_table_deletes_total counter
HELP crpd_lldp_remote_db_table_deletes_total Number of deletions made in the remote database table
crpd_lldp_remote_db_table_deletes_total 0
TYPE crpd_lldp_remote_db_table_drops_total counter
HELP crpd_lldp_remote_db_table_drops_total Number of LLDP frames dropped from the remote database table because of errors
crpd_lldp_remote_db_table_drops_total 0
TYPE crpd_lldp_remote_db_table_ageouts_total counter
HELP crpd_lldp_remote_db_table_ageouts_total Number of remote database table entries that have aged out of the table
crpd_lldp_remote_db_table_ageouts_total 0</code>

Interface Statistics

<code># TYPE crpd_lldp_interface_receive_packets_total counter
HELP crpd_lldp_interface_receive_packets_total Number of LLDP frames received on this interface
crpd_lldp_interface_receive_packets_total{interface="enp1s0"} 1566
TYPE crpd_lldp_interface_receive_unknown_tlvs_total counter
HELP crpd_lldp_interface_receive_unknown_tlvs_total Number of LLDP frames with unsupported content received on this interface
crpd_lldp_interface_receive_unknown_tlvs_total{interface="enp1s0"} 0
TYPE crpd_lldp_interface_receive_errors_total counter
HELP crpd_lldp_interface_receive_errors_total Number of LLDP frames with errors received on this interface
crpd_lldp_interface_receive_errors_total{interface="enp1s0"} 0
TYPE crpd_lldp_interface_receive_tlv_discards_total counter
HELP crpd_lldp_interface_receive_tlv_discards_total Number of LLDP frames received and then discarded on this interface
crpd_lldp_interface_receive_tlv_discards_total{interface="enp1s0"} 0
TYPE crpd_lldp_interface_transmit_packets_total counter
HELP crpd_lldp_interface_transmit_packets_total Number of LLDP frames sent on this interface
crpd_lldp_interface_transmit_packets_total{interface="enp1s0"} 3046</code></td></tr>
</table>

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output

	`# TYPE crpd_lldp_interface_transmit_errors_total counter`
	`# HELP crpd_lldp_interface_transmit_errors_total Number of LLDP frames that were untransmitted on this interface`
	`crpd_lldp_interface_transmit_errors_total{interface="enp1s0"} 1`

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) *(Continued)*

<table>
<tr><th>Group</th><th>Sample Output</th></tr>
<tr><td>Layer 2 Circuit Counters</td><td><pre>
TYPE crpd_l2_circuit_connection_info guage
HELP crpd_l2_circuit_connection_info Details about layer 2 circuit
crpd_l2_circuit_connection_info{neighbor_address="10.10.10.3",
interface="ens1f3v0.144", vcid="144", control_word="yes", inbound_label="16",
outbound_label="16"} 1

TYPE crpd_l2_circuit_connection_status   guage
HELP crpd_l2_circuit_connection_status   Layer 2 circuit connection status
crpd_l2_circuit_connection_status{neighbor_address="10.10.10.3",
interface="ens1f3v0.144"} 1

Possible status values include:

Description   Numeric Value
Down   0
Operational   1
Virtual circuit Down   2
Interface h/w not present   3
Call admission control failure   4
TDM incompatible bitrate   5
TDM misconfiguration   6
Standby Connection   7
Static Pseudowire   8
Remote site standby   9
Hot-standby Connection   10
Encapsulation invalid   11
MTU mismatch   12
Encapsulation mismatch   13
Control-word mismatch   14
VLAN id mismatch   15
No outgoing label   16
Intf encaps not CCC/TCC   17
Backup Connection   18
Rcvd cell-bundle size bad   19
Local site signaled down   20
Remote site signaled down   21
Unknown   22

TYPE crpd_l2_circuit_connection_up_transitions_total counter
HELP crpd_l2_circuit_connection_up_transitions_total   Number of times the virtual
circuit came up
</pre></td></tr>
</table>

Table 25: Sample cRPD Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	`crpd_l2_circuit_connection_up_transitions_total{neighbor_address="10.10.10.3", interface="ens1f3v0.144"} 2`

Table 26: Sample cSRX Telemetry Data (Prometheus-based API)

Group	Sample Output

Global IPSec Statistics	`# TYPE csrx_ipsec_esp_encrypted_packets_total counter`
	`# HELP Total number of packets encrypted by the local system across the IPsec tunnel`
	`csrx_ipsec_esp_encrypted_packets_total 7627`
	`# TYPE csrx_ipsec_esp_encrypted_bytes_total counter`
	`# HELP Total number of bytes encrypted by the local system across the IPsec tunnel`
	`csrx_ipsec_esp_encrypted_bytes_total 872655`
	`# TYPE csrx_ipsec_esp_decrypted_packets_total counter`
	`# HELP Total number of packets decrypted by the local system across the IPsec tunnel`
	`csrx_ipsec_esp_decrypted_packets_total 4329`
	`# TYPE csrx_ipsec_esp_decrypted_bytes_total counter`
	`# HELP Total number of bytes decrypted by the local system across the IPsec tunnel`
	`csrx_ipsec_esp_decrypted_bytes_total 293764`
	`# TYPE csrx_ipsec_ah_input_packets_total counter`
	`# HELP Total number of packets received by the local system across the IPsec tunnel`
	`csrx_ipsec_ah_input_packets_total 10`
	`# TYPE csrx_ipsec_ah_input_bytes_total counter`
	`# HELP Total number of bytes received by the local system across the IPsec tunnel`
	`csrx_ipsec_ah_input_bytes_total 8726`
	`# TYPE csrx_ipsec_ah_output_packets_total counter`
	`# HELP Total number of packets transmitted by the local system across the IPsec tunnel`
	`csrx_ipsec_ah_output_packets_total 9`
	`# TYPE csrx_ipsec_ah_output_bytes_total counter`
	`# HELP Total number of bytes transmitted by the local system across the IPsec tunnel`
	`csrx_ipsec_ah_output_bytes_total 6323`
	`# TYPE csrx_ipsec_ah_auth_failures_total counter`
	`# HELP Total number of authentication header (AH) failures. An AH failure occurs when there is a mismatch of the authentication header in a packet transmitted across an IPsec tunnel`
	`csrx_ipsec_ah_auth_failures_total 0`
	`# TYPE csrx_ipsec_bad_headers_total counter`
	`# HELP Total number of invalid headers detected`
	`csrx_ipsec_bad_headers_total 0`
	`# TYPE csrx_ipsec_bad_trailers_total counter`
	`# HELP Total number of invalid trailers detected`
	`csrx_ipsec_bad_trailers_total 0`
	`# TYPE csrx_ipsec_discard_errors_total counter`
	`# HELP Total number of discarded packets detected`
	`csrx_ipsec_discard_errors_total 0`
	`# TYPE csrx_ipsec_esp_decryption_failures_total counter`
	`# HELP total number of ESP decryption errors`
	`csrx_ipsec_esp_decryption_failures_total 0`

Table 26: Sample cSRX Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	# TYPE csrx_ipsec_esp_auth_failures_total counter # HELP Total number of Encapsulation Security Payload (ESP) failures. An ESP failure occurs when there is an authentication mismatch in ESP packets csrx_ipsec_esp_auth_failures_total 0 # TYPE csrx_ipsec_exceeds_tunnel_mtu_total counter # HELP Total number of times the maximum size of transmit packet exceeds the configured tunnel MTU for IPsec tunnels csrx_ipsec_exceeds_tunnel_mtu_total 0 # TYPE csrx_ipsec_invalid_spi_errors_total counter # HELP Total number of invalid SPIs packets detected csrx_ipsec_invalid_spi_errors_total 0 # TYPE csrx_ipsec_replay_errors_total counter # HELP Total number of replay errors. A replay error is generated when a duplicate packet is received within the replay window csrx_ipsec_replay_errors_total 0 # TYPE csrx_ipsec_ts_check_fail_errors_total counter # HELP Total number of TS check fail detected csrx_ipsec_ts_check_fail_errors_total 0
IKE Gateway Statistics	# TYPE csrx_ike_gateway_active_peers gauge # HELP Number of successful IKE negotiations with the remote peers csrx_ike_gateway_active_peers{gateway="gw-1-1-1-1"} 1 # TYPE csrx_ike_gateway_in_progress_peer_negotiations gauge # HELP Number of in progress IKE negotiations with the remote peers csrx_ike_gateway_in_progress_peer_negotiations{gateway="gw-1-1-1-1"} 0 # TYPE csrx_ike_gateway_failed_peer_negotiations_total counter # HELP Number of failed IKE negotiations with the remote peers csrx_ike_gateway_failed_peer_negotiations_total{gateway="gw-1-1-1-1"} 0 # TYPE csrx_ike_gateway_blocked_peer_negotiations_total counter # HELP Number of blocked IKE negotiations with the remote peers csrx_ike_gateway_blocked_peer_negotiations_total{gateway="gw-1-1-1-1"} 0 # TYPE csrx_ike_gateway_backoff_peer_negotiations_total counter # HELP Number of backed off IKE negotiations with the remote peers csrx_ike_gateway_backoff_peer_negotiations_total{gateway="gw-1-1-1-1"} 0

Table 26: Sample cSRX Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
IKE Peer Statistics	# TYPE csrx_ike_peer_ike_sa_negotiated gauge # HELP Total number of negotiated security associations csrx_ike_peer_ike_sa_negotiated{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ipsec_active_tunnels gauge # HELP Total number of active IPsec tunnels csrx_ike_peer_ipsec_active_tunnels{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_sa_init_requests_sent_total counter # HELP Total number of security association initiation requests sent to the remote address csrx_ike_peer_sa_init_requests_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_sa_init_requests_received_total counter # HELP Total number of security association initiation requests received from the remote address csrx_ike_peer_sa_init_requests_received_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_sa_init_responses_received_total counter # HELP Total number of security association responses received from the remote address csrx_ike_peer_sa_init_responses_received_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_sa_init_responses_sent_total counter # HELP Total number of security association responses sent to the remote address csrx_ike_peer_sa_init_responses_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_auth_requests_sent_total counter # HELP Total number of authentication requests sent to the remote address csrx_ike_peer_auth_requests_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_auth_requests_received_total counter # HELP Total number of authentication requests received from the remote address csrx_ike_peer_auth_requests_received_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_auth_responses_received_total counter # HELP Total number of authentication responses received from the remote address csrx_ike_peer_auth_responses_received_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_auth_responses_sent_total counter # HELP Total number of authentication responses sent to the remote address csrx_ike_peer_auth_responses_sent_total{local_address="1.1.1.1",

Table 26: Sample cSRX Telemetry Data (Prometheus-based API) ***(Continued)***

Group	Sample Output
	remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ike_sa_rekey_requests_sent_total counter # HELP Total number of IKE rekey requests sent to the remote address csrx_ike_peer_ike_sa_rekey_requests_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ike_sa_rekey_requests_received_total counter # HELP Total number of IKE rekey requests received from the remote address csrx_ike_peer_ike_sa_rekey_requests_received_total{local_address="1.1.1.1",remote_addr ess="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ike_sa_rekey_responses_received_total counter # HELP Total number of IKE rekey responses received from the remote address csrx_ike_peer_ike_sa_rekey_responses_received_total{local_address="1.1.1.1",remote_add ress="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ike_sa_rekey_responses_sent_total counter # HELP Total number of IKE rekey responses sent to the remote address csrx_ike_peer_ike_sa_rekey_responses_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ipsec_sa_rekey_requests_sent_total counter # HELP Total number of IPSEC rekey requests sent to the remote address csrx_ike_peer_ipsec_sa_rekey_requests_sent_total{local_address="1.1.1.1", remote_address="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ipsec_sa_rekey_requests_received_total counter # HELP Total number of IPSEC rekey requests received from the remote address csrx_ike_peer_ipsec_sa_rekey_requests_received_total{local_address="1.1.1.1",remote_ad dress="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ipsec_sa_rekey_responses_received_total counter # HELP Total number of IPSEC rekey responses received from the remote address csrx_ike_peer_ipsec_sa_rekey_responses_received_total{local_address="1.1.1.1",remote_a ddress="2.2.2.2", routing_instance="default"} 1 # TYPE csrx_ike_peer_ipsec_sa_rekey_responses_sent_total counter # HELP Total number of IPSEC rekey responses sent to the remote address csrx_ike_peer_ipsec_sa_rekey_responses_sent_total{local_address="1.1.1.1",remote_addre ss="2.2.2.2", routing_instance="default"} 1

Telemetry via gNMI

The gNMI protocol defines the `Subscribe` RPC for subscribing to telemetry data. The telemetry collector uses this RPC to request updates from the network device for state and configuration data. Review Subscribing to Telemetry Data using gNMI for more detail.

You can enable support by enabling the gNMI configuration for cRPD and vRouter in the `values.yaml` helm chart. Please review the Customize the Helm Chart topic for your platform for more details. You can also enable gNMI by passing the following arguments to the helm install command while installing JCNR:

```
--set telemetry.gnmi.enable=true
--set vrouter.telemetry.gnmi.enable=true
```

To enable gNMI configuration for cSRX, configure the `values.yaml` helm chart. Please review the Customize the Helm Chart topic for your platform for more details. You can also enable gNMI by passing the following arguments to the helm install command while installing JCNR:

```
--set telemetry.gnmi.enable=true
```

NOTE:

1. External clients can connect to the grpc port 50053 when telemetry.gnmi.enable is set to true in the helmchart.

2. gNMI subscription works with only authenticated requests from external clients, for example:

   ```
   gnmic -a 10.1.1.1:50053 -u root -p my-root-password --insecure sub --stream-mode
   sample -i 10s --path "/junos/system/state/network/in-pkts"
   ```

3. `SAMPLE` and `TARGET_DEFINED` subscription modes are supported in the current release.

4. Targeting of specific instances in a `SubscribeMessage` using key values in paths is not supported in the current release, for example, subscribing to monitoring the metrics of a specific interface using a path such as `/interfaces/interface[name=”enp3s0”]/state/counters`

5. The gNMI debug logs are available at `/var/log/jcnr/na-grpcd`.

The following table lists the supported sensors for telemetry metrics:

Table 27: Supported Sensors for Telemetry Metrics (cRPD)

Sensor Type	Sensor Paths
Interface Sensors	/interfaces/interface[name="*interface_name*"]/config/description /interfaces/interface[name="*interface_name*"]/ethernet/mac-address /interfaces/interface[name="*interface_name*"]/state/admin-status /interfaces/interface[name="*interface_name*"]/state/oper-status /interfaces/interface[name="*interface_name*"]/state/counters/in-octets /interfaces/interface[name="*interface_name*"]/state/counters/in-pkts /interfaces/interface[name="*interface_name*"]/state/counters/out-octets /interfaces/interface[name="*interface_name*"]/state/counters/out-pkts /interfaces/interface[name="*interface_name*"]/state/counters/in-errors /interfaces/interface[name="*interface_name*"]/state/counters/out-errors /interfaces/interface[name="*interface_name*"]/state/counters/in-port-errors /interfaces/interface[name="*interface_name*"]/state/counters/out-port-errors /interfaces/interface[name="*interface_name*"]/state/counters/in-device-errors /interfaces/interface[name="*interface_name*"]/state/counters/out-device-errors /interfaces/interface[name="*interface_name*"]/state/counters/drops /interfaces/interface[name="*interface_name*"]/state/counters/discards
L3/L4 Access List (Firewall Filter) Counters	/junos/firewall_stats[family="*family*", name="*filter_name*"]/state/counter_stats[name="*counter_name*"]/packets /junos/firewall_stats[family="*family*", name="*filter_name*"]/state/counter_stats[name="*counter_name*"]/bytes
ARP Sensors	/arp-information/ipv4/neighbors/neighbor[ip="*ip_address*"]/state/link-layer-address /arp-information/ipv4/neighbors/neighbor[ip="*ip_address*"]/state/interface-name /arp-information/ipv4/neighbors/neighbor[ip="*ip_address*"]/state/origin

Table 27: Supported Sensors for Telemetry Metrics (cRPD) ***(Continued)***

Sensor Type	Sensor Paths
MAC Learning Sensors	/network-instances/network-instance/fdb/mac-table/entry[vlan=*vlan_id*, mac-address="*mac_address*"]/state/entry-type /network-instances/network-instance/fdb/mac-table/entry[vlan=*vlan_id*, mac-address="*mac_address*"]/mac-address /network-instances/network-instance/fdb/mac-table/entry[vlan=*vlan_id*, mac-address="*mac_address*"]/interface/interface-ref/state/interface /network-instances/network-instance/fdb/mac-table/entry[vlan=*vlan_id*, mac-address="*mac_address*"]/vlan

Table 27: Supported Sensors for Telemetry Metrics (cRPD) ***(Continued)***

Sensor Type	Sensor Paths
TWAMP Sensors	/junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/avg-delay /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/jitter-delay /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/max-delay /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/min-delay /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/samples /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/stddev-delay /junos/twamp/client/probe-test-results[owner="*owner_name*", test-name="*test_name*"]/probe-test-generic-results[results-scope=*enum*]/probe-test-generic-measurements[probe-measurement-type=*enum*]/sum-delay **NOTE**: results-scope *enum* can be one of the following: [CURRENT_TEST|LAST_TEST|MOVING_AVERAGE_TEST|ALL_TESTS] probe-measurement-type *enum* can be one of the following: [ROUND_TRIP_TIME | EGRESS_DELAY | INGRESS_DELAY | POSITIVE_RTT_JITTER | NEGATIVE_RTT_JITTER | POSITIVE_EGRESS_JITTER | NEGATIVE_EGRESS_JITTER | POSITIVE_INGRESS_JITTER | NEGATIVE_INGRESS_JITTER]

Table 27: Supported Sensors for Telemetry Metrics (cRPD) ***(Continued)***

Sensor Type	Sensor Paths
Routing Sensors	/junos/routing/route-tables/route-table[name="*routing_table_name*"]/routes/route[prefix="*prefix*"]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/interface /junos/routing/route-tables/route-table[name="*routing_table_name*"]/routes/route[prefix="*prefix*"]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/next-hop /junos/routing/route-tables/route-table[name="*routing_table_name*"]/routes/route[prefix="*prefix*"]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/protocol /junos/routing/route-tables/route-table[name="*routing_table_name*"]/routes/route[prefix="*prefix*"]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/state/age /junos/routing/route-tables/route-table[name="*routing_table_name*"]/route[prefix="*prefix*]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/state/active For Layer 2 circuit only: /junos/routing/route-tables/route-table[name="*routing_table_name*"]/route[prefix="*prefix*]/entries/entry[protocol="*protocol*",next-hop="*ip_addr*",interface="*interface_name*"]/next-hop-mpls-label

Table 27: Supported Sensors for Telemetry Metrics (cRPD) *(Continued)*

Sensor Type	Sensor Paths
LLDP Sensors	/lldp /lldp/state /lldp/state/enabled /lldp/state/hello-timer /lldp/state/system-name /lldp/state/system-description /lldp/state/chassis-id /lldp/state/chassis-id-type /lldp/state/loc-port-id-type /lldp/state/counters /lldp/state/counters/frame-in /lldp/state/counters/frame-out /lldp/state/counters/frame-error-in /lldp/state/counters/frame-discard /lldp/state/counters/tlv-discard /lldp/state/counters/tlv-unknown /lldp/state/counters/last-clear /lldp/state/counters/tlv-accepted /lldp/state/counters/entries-aged-out /lldp/state/suppress-tlv-advertisement /lldp/interfaces /lldp/interfaces/interface /lldp/interfaces/interface/name /lldp/interfaces/interface/state /lldp/interfaces/interface/state/name /lldp/interfaces/interface/state/enabled /lldp/interfaces/interface/state/counters /lldp/interfaces/interface/state/counters/frame-in

Table 27: Supported Sensors for Telemetry Metrics (cRPD) *(Continued)*

Sensor Type	Sensor Paths
	/lldp/interfaces/interface/state/counters/frame-out /lldp/interfaces/interface/state/counters/frame-error-in /lldp/interfaces/interface/state/counters/frame-discard /lldp/interfaces/interface/state/counters/tlv-discard /lldp/interfaces/interface/state/counters/tlv-unknown /lldp/interfaces/interface/state/counters/last-clear /lldp/interfaces/interface/state/counters/frame-error-out /lldp/interfaces/interface/state/loc-port-id /lldp/interfaces/interface/state/loc-port-description /lldp/interfaces/interface/neighbors /lldp/interfaces/interface/neighbors/neighbor /lldp/interfaces/interface/neighbors/neighbor/id /lldp/interfaces/interface/neighbors/neighbor/state /lldp/interfaces/interface/neighbors/neighbor/state/system-name /lldp/interfaces/interface/neighbors/neighbor/state/system-description /lldp/interfaces/interface/neighbors/neighbor/state/chassis-id /lldp/interfaces/interface/neighbors/neighbor/state/chassis-id-type /lldp/interfaces/interface/neighbors/neighbor/state/id /lldp/interfaces/interface/neighbors/neighbor/state/age /lldp/interfaces/interface/neighbors/neighbor/state/last-update /lldp/interfaces/interface/neighbors/neighbor/state/port-id /lldp/interfaces/interface/neighbors/neighbor/state/port-id-type /lldp/interfaces/interface/neighbors/neighbor/state/port-description /lldp/interfaces/interface/neighbors/neighbor/state/management-address /lldp/interfaces/interface/neighbors/neighbor/state/management-address-type lldp/interfaces/interface/neighbors/neighbor/custom-tlvs lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/type

Table 27: Supported Sensors for Telemetry Metrics (cRPD) *(Continued)*

Sensor Type	Sensor Paths
	/lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/oui /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/oui-subtype /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/state /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/state/type /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/state/oui /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/state/oui-subtype /lldp/interfaces/interface/neighbors/neighbor/custom-tlvs/tlv/state/value /lldp/interfaces/interface/neighbors/neighbor/capabilities /lldp/interfaces/interface/neighbors/neighbor/capabilities/capability /lldp/interfaces/interface/neighbors/neighbor/capabilities/capability/name /lldp/interfaces/interface/neighbors/neighbor/capabilities/capability/state /lldp/interfaces/interface/neighbors/neighbor/capabilities/capability/state/name /lldp/interfaces/interface/neighbors/neighbor/capabilities/capability/state/enabled
Layer 2 Circuit Sensors	/junos/l2-circuit-connections/l2-circuit-connection[neighbor-address="*ip*", local-interface="*name*"]/state/status /junos/l2-circuit-connections/l2-circuit-connection[neighbor-address="*ip*", local-interface="*name*"]/state/vcid /junos/l2-circuit-connections/l2-circuit-connection[neighbor-address="*ip*", local-interface="*name*"]/state/inbound-label /junos/l2-circuit-connections/l2-circuit-connection[neighbor-address="*ip*", local-interface="*name*"]/state/outbound-label /junos/l2-circuit-connections/l2-circuit-connection[neighbor-address="*ip*", local-interface="*name*"]/state/up-transitions

To view additional supported sensors for cRPD please review—Telemetry Information on cRPD.

Table 28: Supported Sensors for Telemetry Metrics (vRouter)

Sensor Type	Sensor Paths

Interface Sensors	`/interfaces/interface[name="`*interface_name*`"]/state/counters/in-octets`
	`/interfaces/interface[name="`*interface_name*`"]/state/counters/in-pkts`
	`/interfaces/interface[name="`*interface_name*`"]/state/counters/out-octets`
	`/interfaces/interface[name="`*interface_name*`"]/state/counters/out-pkts`

Table 28: Supported Sensors for Telemetry Metrics (vRouter) ***(Continued)***

Sensor Type	Sensor Paths
Interface VLAN Sensors	/interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-broadcast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-broadcast-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-broadcast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-broadcast-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-flooded-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-flooded-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-multicast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-multicast-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-multicast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-multicast-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-unicast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/in-unicast-pkts /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-unicast-octets /interfaces/interface[name="*interface_name*"]/subinterfaces/subinterface[index=*vlanID*]/vlan/state/counters/out-unicast-pkts **NOTE**: Interface VLAN sensors are not supported for L3 VLAN Sub-Interfaces.

Table 28: Supported Sensors for Telemetry Metrics (vRouter) ***(Continued)***

Sensor Type	Sensor Paths
Global vRouter Sensors	/junos/system/state/network/in-octets /junos/system/state/network/in-pkts /junos/system/state/network/out-octets /junos/system/state/network/out-pkts /junos/system/state/network/flows /junos/system/state/network/active-flows /junos/system/state/network/aged-flows /junos/system/state/network/hold-flows /junos/system/state/network/drop-pkts /junos/system/state/network/checksum-err-drop-pkts /junos/system/state/network/flow-no-memory-drop-pkts /junos/system/state/network/clone-fail-drop-pkts /junos/system/state/network/discard-drop-pkts /junos/system/state/network/drop-new-flow-drop-pkts /junos/system/state/network/drop-pkts-drop-pkts /junos/system/state/network/duplicated-drop-pkts /junos/system/state/network/flow-action-drop-drop-pkts /junos/system/state/network/flow-action-invalid-drop-pkts /junos/system/state/network/flow-evict-drop-pkts /junos/system/state/network/flow-invalid-protocol-drop-pkts /junos/system/state/network/flow-nat-no-rflow-drop-pkts /junos/system/state/network/flow-no-memory-drop-pkts /junos/system/state/network/flow-queue-limit-exceeded-drop-pkts /junos/system/state/network/flow-table-full-drop-pkts

Table 28: Supported Sensors for Telemetry Metrics (vRouter) ***(Continued)***

Sensor Type	Sensor Paths
	/junos/system/state/network/flow-unusable-drop-pkts /junos/system/state/network/frag-err-drop-pkts /junos/system/state/network/fragment-queue-fail-drop-pkts /junos/system/state/network/head-alloc-fail-drop-pkts /junos/system/state/network/icmp-error-drop-pkts /junos/system/state/network/interface-drop-drop-pkts /junos/system/state/network/interface-rx-discard-drop-pkts /junos/system/state/network/interface-tx-discard-drop-pkts /junos/system/state/network/invalid-arp-drop-pkts /junos/system/state/network/invalid-if-drop-pkts /junos/system/state/network/invalid-label-drop-pkts /junos/system/state/network/invalid-mcast-source-drop-pkts /junos/system/state/network/invalid-nh-drop-pkts /junos/system/state/network/invalid-packet-drop-pkts /junos/system/state/network/invalid-protocol-drop-pkts /junos/system/state/network/invalid-source-drop-pkts /junos/system/state/network/invalid-vnid-drop-pkts /junos/system/state/network/l2-no-route-drop-pkts /junos/system/state/network/mcast-clone-fail-drop-pkts /junos/system/state/network/mcast-df-bit-drop-pkts /junos/system/state/network/misc-drop-pkts /junos/system/state/network/no-fmd-drop-pkts /junos/system/state/network/no-frag-entry-drop-pkts /junos/system/state/network/no-memory-drop-pkts

Table 28: Supported Sensors for Telemetry Metrics (vRouter) *(Continued)*

Sensor Type	Sensor Paths
	/junos/system/state/network/nowhere-to-go-drop-pkts /junos/system/state/network/pcow-fail-drop-pkts /junos/system/state/network/pull-drop-pkts /junos/system/state/network/push-drop-pkts /junos/system/state/network/rewrite-fail-drop-pkts /junos/system/state/network/trap-no-if-drop-pkts /junos/system/state/network/trap-original-drop-pkts /junos/system/state/network/ttl-exceeded-drop-pkts /junos/system/state/network/vlan-fwd-enq-drop-pkts /junos/system/state/network/vlan-fwd-tx-drop-pkts

Table 29: Supported Sensors for Telemetry Metrics (cSRX)

Sensor Type	Sensor Paths
IPSec Sensors	/junos/security/ipsec/counters/esp-encrypted-packets /junos/security/ipsec/counters/esp-encrypted-bytes /junos/security/ipsec/counters/esp-decrypted-packets /junos/security/ipsec/counters/esp-decrypted-bytes /junos/security/ipsec/counters/ah-input-packets /junos/security/ipsec/counters/ah-input-bytes /junos/security/ipsec/counters/ah-output-packets /junos/security/ipsec/counters/ah-output-bytes /junos/security/ipsec/counters/ah-output-bytes /junos/security/ipsec/counters/ah-auth-failures /junos/security/ipsec/counters/bad-headers /junos/security/ipsec/counters/bad-trailers /junos/security/ipsec/counters/discard-errors /junos/security/ipsec/counters/esp-auth-failures /junos/security/ipsec/counters/esp-decryption-failures /junos/security/ipsec/counters/exceeds-tunnel-mtu /junos/security/ipsec/counters/invalid-spi-errors /junos/security/ipsec/counters/replay-errors /junos/security/ipsec/counters/ts-check-fail-errors /junos/security/ike-gateways/ike-gateway[name="*gateway_name*"]/counters/active-peers /junos/security/ike-gateways/ike-gateway[name="*gateway_name*"]/counters/in-progress-peer-negotiations /junos/security/ike-gateways/ike-gateway[name="*gateway_name*"]/counters/failed-peer-negotiations

Table 29: Supported Sensors for Telemetry Metrics (cSRX) ***(Continued)***

Sensor Type	Sensor Paths
	/junos/security/ike-gateways/ike-gateway[name="*gateway_name*"]/counters/blocked-peer-negotiations /junos/security/ike-gateways/ike-gateway[name="*gateway_name*"]/counters/backoff-peer-negotiations /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/active-peers /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/in-progress-peer-negotiations /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/failed-peer-negotiations /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/blocked-peer-negotiations /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/backoff-peer-negotiations /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ipsec-active-tunnels /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ike-sa-negotiated /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/sa-init-requests-sent /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/sa-init-responses-received

Table 29: Supported Sensors for Telemetry Metrics (cSRX) ***(Continued)***

Sensor Type	Sensor Paths
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/sa-init-responses-sent
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/auth-requests-received
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/auth-requests-sent
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/auth-responses-received
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/auth-responses-sent
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ike-sa-rekey-requests-received
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ike-sa-rekey-requests-sent
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ike-sa-rekey-responses-received
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ike-sa-rekey-responses-sent
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ipsec-sa-rekey-requests-received
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ipsec-sa-rekey-requests-sent

Table 29: Supported Sensors for Telemetry Metrics (cSRX) ***(Continued)***

Sensor Type	Sensor Paths
	/junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ipsec-sa-rekey-responses-received /junos/security/ike-peers/ike-peer[local-address="*local_address*",remote-address="*remote_address*",routing-instance="*routing_instance_name*"]/counters/ipsec-sa-rekey-responses-sent

Troubleshooting

You can view the cRPD telemetry logs under `/var/log/jcnr/na-grpcd` on your Cloud-Native Router server. You can view the Vrouter telemetry logs using the `kubectl -n contrail logs -c contrail-vrouter-telemetry-exporter` *vRouter_pod_name* command. You can view the cSRX telemetry logs using the `kubectl -n jcnr logs -c csrx-telemetry-exporter` *csrx_pod_name*The supported logging levels are `error`, `warning`, `info`, `debug`, `trace` and `verbose`. The default logging level is set to `debug` and can be modified in the helm chart.

Logging and Notifications

IN THIS SECTION

Read this topic to learn about logging and notification functions in Juniper Cloud-Native Router. We discuss the location of log files, what you can log, and various log levels. You can also learn about the available notifications and how the notifications are implemented in the Cloud-Native Router.

Logging

The system stores log files from all the Cloud-Native Router pods and containers in the **log_path** directory. You can configure the location of the log files at the deployment time by retaining or changing the value of the **log_path** key in the `values.yaml` helm chart. By default, the location of the log files is **/var/log/jcnr**.

The Juniper Cloud-Native Router pods and containers use syslog as their logging mechanism. The syslog-ng pod stores event notification data in JSON format on the host server. By default, the file location is **/var/log/jcnr** and the filename is **jcnr_notifications.json**. You can change the location and filename by changing the value of the **syslog_notifications** key in the `values.yaml` helm chart when installing the Cloud-Native Router. You can disable syslog logging by configuring the `installSyslog` parameter as false in the `values.yaml` helm chart. Please review the *Customizing Cloud-Native Router Helm Chart* for more details.

When you use the default file locations, the **/var/log/jcnr** directory displays the following files:

```
[root@jcnr-01 jcnr]# ls
action.log                    contrail-vrouter-dpdk-init.log  filter
l2cos.log      __policy_names_rpdc__
contrail-vrouter-agent.log    contrail-vrouter-dpdk.log       filter.log
license        mgd-api
__policy_names_rpdn__         cos                             jcnr-cni.log
messages       mosquitto
vrouter-kernel-init.log       cscript.log                     jcnr_notifications.json
messages.0.gz  na-grpcd
```

NOTE: The host server must manage the log rotation for the **contrail-vrouter-dpdk.log** and the **jcnr-cni.log** files.

Notifications

The syslog-ng pod continuously monitors the preceding log files for notification events such as interface up, interface down, interface add, and so on. When these events appear in a log file, syslog-ng converts the log events into notification events and stores the events in JSON format within the **syslog_notifications** file configured in the **values.yaml** file.

Here is a sample of syslog-ng notifications:

Table 30: Supported Notifications

Notification	Source Pod
License Near Expiry	cRPD
License Expired	cRPD
License Invalid	cRPD
License OK	cRPD
License Grace Period	cRPD
License Not Present	cRPD
Cloud-Native Router Init Success	Deployer
Cloud-Native Router Init Failure	Deployer
Cloud-Native Router Graceful Shutdown Request	Deployer
Cloud-Native Router Graceful Shutdown Complete	Deployer
Cloud-Native Router Graceful Shutdown Failure	Deployer
Cloud-Native Router Restart	Deployer
Cloud-Native Router Upgrade Success	Deployer
Cloud-Native Router Upgrade Failure	Deployer
Upstream Fabric Bond Member Link Up	vRouter
Upstream Fabric Bond Member Link Down	vRouter
Upstream Fabric Bond Link Up	vRouter

Table 30: Supported Notifications *(Continued)*

Notification	Source Pod
Upstream Fabric Bond Link Down	vRouter
Upstream Fabric Bond Link Switchover	vRouter
Downstream Fabric Link Up	vRouter
Downstream Fabric Link Down	vRouter
Appliance Link Up	vRouter
Appliance Link Down	vRouter
Any Cloud-Native Router Application Critical Errors	vRouter
Any Cloud-Native Router Application Warnings	vRouter
Any Cloud-Native Router Application Info	vRouter
Cloud-Native Router Rate Limits Reached	vRouter
Cloud-Native Router MAC Table Limit Reached	vRouter
Cloud-Native Router CLI Start	cRPD or vRouter-Agent
Cloud-Native Router CLI Stop	cRPD or vRouter-Agent
Cloud-Native Router Kernel App Interface Up	vRouter
Cloud-Native Router Kernel App Interface Down	vRouter
Cloud-Native Router Virtio User Interface Up	vRouter
Cloud-Native Router Virtio User Interface Down	vRouter

Table 30: Supported Notifications *(Continued)*

Notification	Source Pod
Purel2 Macmove Loop Detected	vRouter

6

CHAPTER

Troubleshooting

IN THIS CHAPTER

Troubleshoot using the vRouter CLI

IN THIS SECTION

Read this topic to learn about the various troubleshooting commands available in the vRouter CLI including `vif`, `purel2cli`, `dpdkinfo`, `flow`, `rt`, `nh` commands.

Accessing the vRouter CLI

Refer to "Access vRouter CLI" on page 331 to learn how to access the vRouter CLI.

Troubleshooting via the vRouter CLI

You can run commands in the CLI to learn about the state of the vRouter.

Verify vRouter Interfaces via the `vif` Command

The command shown below allows you to see which interfaces are present on the vRouter:

```
vif --list
Vrouter Operation Mode: PureL2
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
```

```
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled

vif0/0      Socket: unix
            Type:Agent HWaddr:00:00:5e:00:01:00
            Vrf:65535 Flags:L2 QOS:-1 Ref:3
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:11  bytes:4169 errors:0
            Drops:0

vif0/1      PCI: 0000:00:00.0 (Speed 25000, Duplex 1)
            Type:Physical HWaddr:46:37:1f:de:df:bc
            Vrf:65535 Flags:L2Vof QOS:-1 Ref:8
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond0  Status: UP  Driver: net_bonding
            Slave Interface(0): 0000:3b:02.0  Status: UP  Driver: net_iavf
            Slave Interface(1): 0000:3b:02.1  Status: UP  Driver: net_iavf
            Vlan Mode: Trunk  Vlan: 100 200 300 700-705
            RX packets:0  bytes:0 errors:0
            TX packets:378  bytes:81438 errors:0
            Drops:0

vif0/2      PCI: 0000:3b:0a.0 (Speed 25000, Duplex 1)
            Type:Workload HWaddr:ba:69:c0:b7:1f:ba
            Vrf:0 Flags:L2Vof QOS:-1 Ref:7
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:3b:0a.0  Status: UP  Driver: net_iavf
            Vlan Mode: Access  Vlan Id: 700  OVlan Id: 700
            RX packets:378  bytes:81438 errors:2
            TX packets:0  bytes:0 errors:0
            Drops:391
```

View the running configuration of the vRouter

To see the status of the vRouter, enter the following command in the vRouter CLI:

```
[root@jcnr-01 /]# ps -eaf | grep vrouter-dpdk
root         116      90 99 Mar30 ?        118-08:05:37 /contrail-vrouter-dpdk --no-daemon --
socket-mem=1024 1024
--allow=0000:5a:02.0 --
vdev=eth_bond_bond0,mode=1,socket_id=0,mac=3a:1a:b7:86:1c:4f,primary=0000:5a:02.0,
```

```
slave=0000:5a:02.0 --l2_table_size=10240 --yield_option 0 --ddp --l2_mode
root     1134749 1134365  0 16:41 pts/0    00:00:00 grep --color=auto vrouter-dpdk
```

The output contains several elements.

Table 31: vRouter Status Attributes

Flag	Meaning
`--l2_mode`	The vRouter is running in L2 mode.
`--l2_table_size`	The current number of entries in the MAC table. The default size is 10240 entries.
`--allow=<PCI Id>`	The PCI ID of fabric and fabric workload interfaces. More than one ID can appear in the output. These IDs serve as an allowlist.
`--ddp`	Enable Intel DDP support. We enable DDP by default in the **values.yaml** file in the vRouter. **NOTE**: The Intel XL710 NIC does not support DDP.

View L2 Configuration and Statistics via the `purel2cli` Command

The `purel2cli` command is a useful utility to view the Cloud-Native Router L2 configuration and statistics. Start by using the **`purel2cli --help`** command.

```
[root@jcnr-01 /]# purel2cli --help
Usage: purel2cli [--mac show]
           [--mac-move-table show]
           [--vlan show]
           [--vlan get <VLAN_ID>]
           [--acl show <VLAN_ID>]
           [--acl reset-counters <VLAN_ID>]
           [--l2stats get <VIF_ID> <VLAN_ID>]
           [--clear VLAN_ID]
           [--qos classifier/re-write/scheduler <NAME>]
           [--qos cla/rw/sch <NAME>]
           [--nolocal show]
```

```
            [--nolocal get <VLAN_ID>]
            [--sock-dir <sock dir>]
            [--help]
```

The **purel2cli --mac show** command shows the MAC addresses that the vRouter has dynamically learned.

```
purel2cli --mac show
==================================================
||  MAC             vlan      port      hit_count||
==================================================
00:01:01:01:01:03  1221      2          1101892
00:01:01:01:01:02  1221      2          1101819
00:01:01:01:01:04  1221      2          1101863
00:01:01:01:01:01  1221      2          1101879
5a:4c:4c:75:90:fe  1250      5          12
Total Mac entries 5
```

The **purel2cli --mac-move-table show** command displays the MAC_MOVE table with MAC addresses, VLAN IDs, hit counts and last update timestamps. The vRouter mitigates L2 loop caused by frequent MAC address movements between ports using the MAC_MOVE table. The MAC_MOVE table tracks MAC address movements and implements loop detection logic based on hit counts and timestamps. When a loop is detected, the vRouter logs a Syslog event and shuts down the affected ports.

```
purel2cli --mac-move-table show
==================================================
||  MAC             vlan      hit_count     timestamp||
==================================================
00:01:01:01:01:03  1221      46            1731038878
Total Mac entries 1
```

The **purel2cli --vlan show** command shows the VLANs and associated ports.

```
purel2cli --vlan show
VLAN      PORT
===============
1201      1,2,3,4,
1202      1,2,3,4,
1203      1,2,3,4,
1204      1,2,3,4,
1205      1,2,3,4,
```

You can also issue the `purel2cli --vlan get` command to get more details about the VLAN.

```
purel2cli --vlan get <vlan-id>
```

Issue the `purel2cli --l2stats` command to view L2 statistics. For example:

```
purel2cli --l2stats get <virtual_interface_ID> <VLAN_ID>
```

```
purel2cli --l2stats get 2 1221
Vlan id count: 1
--------------------------------------------------------------------------------
Statistics for vif 2 vlan 1221
--------------------------------------------------------------------------------
                Rx Pkts          Rx Bytes           Tx Pkts            Tx Bytes
Unicast        245344824       48152682842          835552           1667761792
Broadcast              0                 0               0                    0
Multicast              0                 0               0                    0
Flood                  0                 0               0                    0
--------------------------------------------------------------------------------
```

```
purel2cli --clear '*'
```

```
purel2cli --clear 100
```

Table 32: `purel2cli` Command Options for L2 Statistics

Sample Command	Function
`purel2cli --l2stats get '*' '*'`	Get statistics for all virtual interfaces (vif) and all VLAN IDs.
`purel2cli --l2stats get '*' 100`	Get statistics for all vif that are part of VLAN 100
`purel2cli --l2stats get 1 '*'`	Get statistics for all VLANs for which interface 1 is a member

Table 32: `purel2cli` Command Options for L2 Statistics *(Continued)*

Sample Command	Function
`purel2cli --l2stats get 1 100`	Get statistics for interface 1 and VLAN 100

The command shows the VLAN to port mapping in the vRouter.You can use the command to see the bridge domain table entry for a specific VLAN: There are several variations of the command that allow you to display and filter L2 statistics in the vRouter. The base form of the command is: . The table below shows the available command options and what they do. It also provides a sample output using one of the options:The following command is an example of the L2 statistics for interface 2 and VLAN 1221:You can clear the statistics from the vRouter with the purel2cli command in the form: . Clears all statistics from all VLANs in the vRouter. Clears all statistics for VLAN id 100.

Packet Tracing via the `dropstats` Command

The vRouter tracks the packets that it drops and includes the reason for dropping them. The table below shows the common reasons for vRouter to drop a packet. When you execute the **dropstats** command, the vRouter does not show a counter if the count for that counter is 0.

Table 33: Dropstats Counters

Counter Name	Meaning
`L2 bd table drop`	No interfaces in bridge domain
`L2 untag pkt drop`	Untagged packet arrives on trunk or sub-interface
`L2 Invalid Vlan`	Packet VLAN does not match interface VLAN
`L2 Mac Table Full`	No more entries available in the MAC table
`L2 ACL drop`	Packet matched firewall filter (ACL) drop rule
`L2 Src Mac lookup fail`	Unable to match (or learn) the source MAC address

Example output from the **dropstats** command looks like:

```
dropstats
L2 bd table Drop              43
L2 untag pkt drop             716
L2 Invalid Vlan               7288253
Rate limit exceeded           673179706
```

```
L2 Mac Table Full             41398787
L2 ACL drop                   8937037
L2 Src Mac lookup fail        247046
```

View status and statistics of DPDK using the `dpdkinfo` Command

The **dpdkinfo** command provides insight into the status and statistics of DPDK. The **dpdkinfo** command has many options. The following sections describe the available options and the example output from the **dpdkinfo** command. You can run the **dpdkinfo** command only from within the vRouter-agent CLI.

```
dpdkinfo --help
Usage: dpdkinfo [--help]
                 --version|-v                                                   Show DPDK
Version
                 --bond|-b                                                      Show Master/
Slave bond information
                 --lacp|-l     <all/conf>                                       Show LACP
information from DPDK
                 --mempool|-m  <all/<mempool-name>>                             Show Mempool
information
                 --stats|-n    <vif index value>                                Show Stats
information
                 --xstats|-x   <vif index value>                                Show Extended
Stats information
                 --lcore|-c                                                     Show Lcore
information
                 --app|-a                                                       Show App
information
                 --ddp|-d      <list> <list-flow>                        Show DDP information
for X710 NIC
                 --rx_vlan|-z  <value>                                          Show VLan
information
       Optional: --buffsz      <value>                                          Send output
buffer size (less than 1000Mb)
```

The command `dpdkinfo -c` shows the Lcores assigned to DPDK VF fabric interfaces and the queue ID for each interface.

```
dpdkinfo -c
No. of forwarding lcores: 4

Lcore 10:
```

```
    Interface: 0000:18:01.1       Queue ID: 0
    Interface: 0000:18:0d.1       Queue ID: 0
    Interface: 0000:86:00.0       Queue ID: 0

Lcore 11:
    Interface: 0000:18:01.1       Queue ID: 1
    Interface: 0000:18:0d.1       Queue ID: 1
    Interface: 0000:86:00.0       Queue ID: 1

Lcore 12:
    Interface: 0000:18:01.1       Queue ID: 2
    Interface: 0000:18:0d.1       Queue ID: 2
    Interface: 0000:86:00.0       Queue ID: 2

Lcore 13:
    Interface: 0000:18:01.1       Queue ID: 3
    Interface: 0000:18:0d.1       Queue ID: 3
    Interface: 0000:86:00.0       Queue ID: 3
```

The command **dpdkinfo -m all** shows all of the memory pool information.

```
dpdkinfo -m all
---------------------------------------------------
Name            Size    Used    Available
---------------------------------------------------
rss_mempool             16384    1549    14835
frag_direct_mempool     4096    0    4096
frag_indirect_mempool    4096    0    4096
packet_mbuf_pool        8192    2    8190
```

The command **dpdkinfo -n 3** displays statistical information for a specific interface.

```
dpdkinfo -n 3
Interface Info(0000:18:0d.1):
RX Device Packets:6710, Bytes:1367533, Errors:0, Nombufs:0
Dropped RX Packets:0
TX Device Packets:0, Bytes:0, Errors:0
Queue Rx:
      Tx:
      Rx Bytes:
```

```
    Tx Bytes:
    Errors:
```

The command **dpdkinfo -x 3** displays extended statistical information for a specific interface.

```
dpdkinfo -x 3
Driver Name:net_iavf
Interface Info:0000:18:0d.1
Rx Packets:
    rx_good_packets: 6701
    rx_unicast_packets: 0
    rx_multicast_packets: 2987
    rx_broadcast_packets: 3714
    rx_dropped_packets: 0
Tx Packets:
    tx_good_packets: 0
    tx_unicast_packets: 0
    tx_multicast_packets: 0
    tx_broadcast_packets: 0
    tx_dropped_packets: 0
Rx Bytes:
    rx_good_bytes: 1365696
Tx Bytes:
    tx_good_bytes: 0
Errors:
    rx_missed_errors: 0
    rx_errors: 0
    tx_errors: 0
    rx_mbuf_allocation_errors: 0
    inline_ipsec_crypto_ierrors: 0
    inline_ipsec_crypto_ierrors_sad_lookup: 0
    inline_ipsec_crypto_ierrors_not_processed: 0
    inline_ipsec_crypto_ierrors_icv_fail: 0
    inline_ipsec_crypto_ierrors_length: 0
Others:
    inline_ipsec_crypto_ipackets: 0
----------------------------------------------------------------------
```

Display routes and next hops using the `rt` and `nh` Commands for L3 Deployments

Use the `rt` command to display all routes in a VRF. The `nh` command enables you to inspect the next hops that are known by the vRouter. Next hops tell the vRouter the next location to send a packet in the path to its final destination.

For example, for IPv4 traffic:

```
rt --get 172.68.20.2/32 --vrf 4
Match 172.68.20.2/32 in vRouter inet4 table 0/4/unicast
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/4/unicast
Destination           PPL        Flags       Label         Nexthop    Stitched MAC(Index)
172.68.20.2/32          0          LPT         16            193        -
```

```
nh --get 193
Id:193        Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:264        Vrf:0
                Flags:Valid, Policy, MPLSoUDP, Etree Root,
Oif:4 Len:14 Data:88 e6 4b 09 7d 46 40 a6 b7 2c a4 48 08 00 Sip:1.1.1.35 Dip:1.1.24.24
```

For example, for IPv6 traffic:

```
rt --get 2001:172:68:20::/64 --vrf 4 --family inet6
Match 2001:172:68:20::/64 in vRouter inet6 table 0/4/unicast
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet6 routing table 0/4/unicast
Destination           PPL        Flags       Label         Nexthop    Stitched MAC(Index)
2001:172:68:20::/64     0          LPT         16            193        -
```

```
nh --get 193
Id:193        Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:264        Vrf:0
              Flags:Valid, Policy, MPLSoUDP, Etree Root,
Oif:4 Len:14 Data:88 e6 4b 09 7d 46 40 a6 b7 2c a4 48 08 00 Sip:1.1.1.35 Dip:1.1.24.24
```

Display all active flows using the `flow` Command for L3 Deployment

Use the `flow` command to display all active flows in a system. For example:

```
flow -l --match 169.83.47.170:9398
Flow table(size 161218560, entries 629760)

Entries: Created 162630 Added 162614 Deleted 35136 Changed 35202Processed 162630 Used Overflow
entries 0
(Created Flows/CPU: 0 0 0 0 0 0 0 0 0 0 241 546 15 161828)(oflows 0)
```

```
Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead
 Stats:Packets/Bytes

Listing flows matching ([169.83.47.170]:9398)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
   328196<=>524233       169.83.47.170:9398                                  6 (2)
                         172.68.20.20:2159
(Gen: 3, K(nh):206, Action:F, Flags:, TCP:, E:1, QOS:-1, S(nh):206,  Stats:6/360,
 SPort 63929, TTL 0, Sinfo 38.0.0.0)

   524233<=>328196       172.68.20.20:2159                                   6 (2)
                         169.83.47.170:9398
(Gen: 3, K(nh):206, Action:F, Flags:, TCP:, QOS:-1, S(nh):250,  Stats:0/0,
 SPort 60311, TTL 0, Sinfo 0.0.0.0)
```

Troubleshoot using Introspect

IN THIS SECTION

Introspect

For vRouter-agent debugging, we use Introspect. You can access the Introspect data at **http://<host server IP>:8085**. Here is a sample of the Introspect data:

Table 34: Modules shown in contrail-vrouter-agent debug output

Link	and Description
agent.xml	Shows agent operational data. Using this introspect, you can see the list of interfaces, VMs, VNs, VRFs, security groups, ACLs and mirror configurations.
agent_ksync.xml	Shows agent ksync layer for data objects such as interfaces and bridge ports.
agent_profile.xml	shows agent **operdb**, tasks, flows, and statistics summary.
agent_stats_interval.xml	View and set collection period for statistics.
controller.xml	Shows the connection status of the jcnr-controller (cRPD)
cpuinfo.xml	Shows the CPU load and memory usage on the compute node.
ifmap_agent.xml	Shows the current configuration data received from **ifmap**.
kstate.xml	Shows data configured in the vRouter data path.
mac_learning.xml	Shows entries in vRouter-agent MAC learning table.
sandesh_trace.xml	Gives the different agent module traces such as **oper**, **ksync**, **mac learning**, and **grpc**.
sandesh_uve.xml	Lists all the user visible entitities (UVEs) in the vRouter-agent. The UVEs are used for analytics and telemetry.
stats.xml	Shows vRouter-agent slow path statistics such as error packets, trapped packets, and debug statistics.
task.xml	Shows vRouter-agent worker task details.

NOTE: The table shows grouped output. The cloud-native router does not group or sort the output on live systems.

The **http://*host server IP address*:8085** page displays only a list of HTML links.

7

CHAPTER

Appendix

IN THIS CHAPTER

Access cRPD CLI

Use the *configlet resource* to configure the cRPD pods.

You can access the command-line interface (CLI) of the cloud-native router controller by accessing the shell of the running cRPD container to verify or troubleshooting the configuration.

NOTE: The commands below are provided as an example. The cRPD pod name must be replaced from your environment. The command outputs may differ based on your environment.

View the running pods in the cluster:

```
kubectl get pods -A
NAMESPACE         NAME                                         READY   STATUS
RESTARTS         AGE
contrail-deploy   contrail-k8s-deployer-5dff6d8b89-7pt9c       1/1     Running
0                138m
contrail          contrail-tools-p27js                         1/1     Running
0                138m
contrail          jcnr-0-dp-contrail-vrouter-nodes-fslcb       2/2     Running
0                138m
contrail          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-pw9w6 1/1    Running
0                138m
jcnr              jcnr-0-crpd-0                                2/2     Running
0                135m
jcnr              syslog-ng-svzxg                              1/1     Running
0                138m
kube-system       calico-kube-controllers-7675746f76-bqbvc     1/1     Running
0                179d
kube-system       calico-node-pt7vb                            1/1     Running
0                540d
kube-system       coredns-59d6b54d97-2qms8                     1/1     Running
0                540d
kube-system       dns-autoscaler-7944dc7978-9t7zd              1/1     Running
0                540d
kube-system       kube-apiserver-5d8s1-node1                   1/1     Running
0                483d
kube-system       kube-controller-manager-5d8s1-node1          1/1     Running
0                359d
```

```
kube-system       kube-multus-ds-amd64-hhhk9                    1/1     Running
0               126d
kube-system       kube-proxy-nfvrl                              1/1     Running
0               540d
kube-system       kube-scheduler-5d8s1-node1                    1/1     Running
0               359d
kube-system       kube-sriov-cni-ds-amd64-wcsfh                 1/1     Running
0               126d
kube-system       kube-sriov-device-plugin-amd64-b9qvp          1/1     Running
0               223d
kube-system       nodelocaldns-qzjq9                            1/1     Running
0               540d
```

Copy the name of the cRPD pod—`jcnr-0-crpd-0` in this example output . You will use the pod name to connect to the running container's shell.

Connect to the cRPD CLI

Issue the `kubectl exec` command to access the running container's shell:

```
kubectl exec -n <namespace> -it <pod name> --container <container name> -- bash
```

where *<namespace>* identifies the namespace in which the pod is running, *<pod name>* specificies the name of the pod and the *<container name>* specifies the name of the container (to be specified if the pod has more than one container).

The cRPD pod has only one running container. Here is an example command:

```
Defaulted container "kube-crpd-worker" out of: kube-crpd-worker, jcnr-crpd-config (init),
install-cni (init)

===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2020-2022, Juniper Networks, Inc. All rights reserved.
                                                                      <===
root@jcnr-01:/#
```

At this point, you have connected to the shell of the cRPD. Just as with other Junos-based shells, you access the operational mode of the cloud-native router the same way as if you were connected to the console of a physical Junos OS device.

```
root@jcnr-01:/# cli
root@jcnr-cni>
```

Access vRouter CLI

You can access the command-line interface (CLI) of the vRouter by accessing the shell of the running vRouter-agent container.

NOTE: The commands below are provided as an example. The vRouter pod name must be replaced from your environment. The command outputs may differ based on your environment.

List the running pods on the K8s Cluster:

```
kubectl get pods -A
NAMESPACE         NAME                                         READY    STATUS
RESTARTS        AGE
contrail-deploy   contrail-k8s-deployer-5dff6d8b89-7pt9c       1/1      Running
0               138m
contrail          contrail-tools-p27js                         1/1      Running
0               138m
contrail          jcnr-0-dp-contrail-vrouter-nodes-fslcb       2/2      Running
0               138m
contrail          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-pw9w6  1/1    Running
0               138m
jcnr              jcnr-0-crpd-0                                2/2      Running
0               135m
jcnr              syslog-ng-svzxg                              1/1      Running
0               138m
kube-system       calico-kube-controllers-7675746f76-bqbvc     1/1      Running
0               179d
kube-system       calico-node-pt7vb                            1/1      Running
0               540d
```

```
kube-system       coredns-59d6b54d97-2qms8                     1/1     Running
0               540d
kube-system       dns-autoscaler-7944dc7978-9t7zd              1/1     Running
0               540d
kube-system       kube-apiserver-5d8s1-node1                   1/1     Running
0               483d
kube-system       kube-controller-manager-5d8s1-node1          1/1     Running
0               359d
kube-system       kube-multus-ds-amd64-hhhk9                   1/1     Running
0               126d
kube-system       kube-proxy-nfvrl                             1/1     Running
0               540d
kube-system       kube-scheduler-5d8s1-node1                   1/1     Running
0               359d
kube-system       kube-sriov-cni-ds-amd64-wcsfh                1/1     Running
0               126d
kube-system       kube-sriov-device-plugin-amd64-b9qvp         1/1     Running
0               223d
kube-system       nodelocaldns-qzjq9                           1/1     Running
0               540d
```

Copy the name of the vRouter pod—`cjcnr-0-dp-contrail-vrouter-nodes-fslcb` in this example output . You will use the pod name to connect to the running container's shell.

Issue the `kubectl exec` command to access the running container's shell:

```
kubectl exec -n <namespace> -it <pod name> --container <container name> -- bash
```

where *<namespace>* identifies the namespace in which the pod is running, *<pod name>* specificies the name of the pod and the *<container name>* specifies the name of the container (to be specified if the pod has more than one container).

The vRouter pod has three containers. When the container name is not specified, the command will default to the vrouter-agent container shell. Here is an example:

```
[root@jcnr-01]# kubectl exec -n contrail -it jcnr-0-dp-contrail-vrouter-nodes-fslcb -- bash
Defaulted container "contrail-vrouter-agent" out of: contrail-vrouter-agent, contrail-vrouter-
telemetry-exporter, contrail-init (init)
[root@jcnr-01 /]#
```

At this point, you have connected to the vRouter's CLI.

Juniper Technology Previews (Tech Previews)

Tech Previews enable you to test functionality and provide feedback during the development process of innovations that are not final production features. The goal of a Tech Preview is for the feature to gain wider exposure and potential full support in a future release. Customers are encouraged to provide feedback and functionality suggestions for a Technology Preview feature before it becomes fully supported.

Tech Previews may not be functionally complete, may have functional alterations in future releases, or may get dropped under changing markets or unexpected conditions, at Juniper's sole discretion. Juniper recommends that you use Tech Preview features in non-production environments only.

Juniper considers feedback to add and improve future iterations of the general availability of the innovations. Your feedback does not assert any intellectual property claim, and Juniper may implement your feedback without violating your or any other party's rights.

These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. Certain features may have reduced or modified security, accessibility, availability, and reliability standards relative to General Availability software. Tech Preview features are not eligible for P1/P2 JTAC cases, and should not be subject to existing SLAs or service agreements.

For additional details, please contact Juniper Support or your local account team.