

Accessing the Networking Stack

The Cisco IOS XR Software serves as a networking stack for communication. This section explains how applications on IOS XR can communicate with internal processes, and with servers or outside devices.

- Packet I/O on IOS XR, on page 1
- Communication Outside Cisco IOS XR, on page 15
- East-West Communication for Third-Party Applications, on page 17
- Telemetry Data Export from Line Card Port via User Defined VRF, on page 19
- MPLS Explicit Null Label for IPv6 Data Packets, on page 20

Packet I/O on IOS XR

This section illustrates how, with the Packet I/O functionality, you can use Linux applications to manage communication with the IOS XR interfaces. It describes how the OS environment must be set up to establish packet I/O communication with hosted applications.

Exposed IOS-XR Interfaces in Linux

The secondary IPv4 addresses that are configured for an XR interface are now synchronized into the Linux operating system automatically. With this secondary IPv4 address synchronization, the third party applications that are deployed on Cisco IOS XR can now use the secondary IPv4 addresses. Prior to this release, only primary IPv4 addresses were supported and the secondary IPv4 addresses had to be configured manually in the Linux operating system.

Exposed XR interfaces (EXIs) and address-only interfaces support secondary IPv4 address synchronization:

- EXIs have secondary IP addresses added to their corresponding Linux interface
- Address-only interfaces have secondary IP addresses added to the Linux loopback device. For additional information on address-only interfaces, see show linux networking interfaces address-only.

The restrictions of secondary IPv4 addresses synchronization are:

- Secondary IPv4 addresses are not synchronized from Linux to XR for Linux-managed interfaces.
- The **ifconfig** Linux command only displays the first configured IPv4 address. To view the complete list of IPv4 addresses, use the **ip addr show** Linux command.

For additional information on secondary IPv4 addresses, see ipv4 address (network).

You can run **bash** commands at the IOS XR router prompt to view the interfaces and IP addresses stored in global VRF. When you access the Cisco IOS XR Linux shell, you directly enter the global VRF.

SUMMARY STEPS

1. From your Linux box, access the IOS XR console through SSH, and log in.
2. View the ethernet interfaces on IOS XR.
3. Check the IP and MAC addresses of the interface that is in `Up` state. Here, interfaces `HundredGigE0/0/0/24` and `MgmtEth0/RP0/CPU0/0` are in the `Up` state.
4. Verify that the bash command runs in global VRF to view the network interfaces.
5. Access the Linux shell.
6. (Optional) View the IP routes used by the `to_xr` interfaces.

DETAILED STEPS

Procedure

Step 1 From your Linux box, access the IOS XR console through SSH, and log in.

Example:

```
cisco@host:~$ ssh root@192.168.122.188
root@192.168.122.188's password:
Router#
```

Step 2 View the ethernet interfaces on IOS XR.

Example:

```
Router#show ip interface brief
Interface IP-Address Status Protocol Vrf-Name
FourHundredGigE0/0/0/0 unassigned Shutdown Down default
FourHundredGigE0/0/0/1 unassigned Shutdown Down default
FourHundredGigE0/0/0/2 unassigned Shutdown Down default
FourHundredGigE0/0/0/3 unassigned Shutdown Down default
FourHundredGigE0/0/0/4 unassigned Shutdown Down default
FourHundredGigE0/0/0/5 unassigned Shutdown Down default
FourHundredGigE0/0/0/6 unassigned Shutdown Down default
FourHundredGigE0/0/0/7 unassigned Shutdown Down default
FourHundredGigE0/0/0/8 unassigned Shutdown Down default
FourHundredGigE0/0/0/9 unassigned Shutdown Down default
FourHundredGigE0/0/0/10 unassigned Shutdown Down default
FourHundredGigE0/0/0/11 unassigned Shutdown Down default
FourHundredGigE0/0/0/12 unassigned Shutdown Down default
FourHundredGigE0/0/0/13 unassigned Shutdown Down default
FourHundredGigE0/0/0/14 unassigned Shutdown Down default
FourHundredGigE0/0/0/15 unassigned Shutdown Down default
FourHundredGigE0/0/0/16 unassigned Shutdown Down default
FourHundredGigE0/0/0/17 unassigned Shutdown Down default
FourHundredGigE0/0/0/18 unassigned Shutdown Down default
FourHundredGigE0/0/0/19 unassigned Shutdown Down default
FourHundredGigE0/0/0/20 unassigned Shutdown Down default
FourHundredGigE0/0/0/21 unassigned Shutdown Down default
FourHundredGigE0/0/0/22 unassigned Shutdown Down default
FourHundredGigE0/0/0/23 unassigned Shutdown Down default
HundredGigE0/0/0/24 10.1.1.10 Up Up default
HundredGigE0/0/0/25 unassigned Shutdown Down default
HundredGigE0/0/0/26 unassigned Shutdown Down default
```

```
HundredGigE0/0/0/27 unassigned Shutdown Down default
HundredGigE0/0/0/28 unassigned Shutdown Down default
HundredGigE0/0/0/29 unassigned Shutdown Down default
HundredGigE0/0/0/30 unassigned Shutdown Down default
HundredGigE0/0/0/31 unassigned Shutdown Down default
HundredGigE0/0/0/32 unassigned Shutdown Down default
HundredGigE0/0/0/33 unassigned Shutdown Down default
HundredGigE0/0/0/34 unassigned Shutdown Down default
HundredGigE0/0/0/35 unassigned Shutdown Down default
MgmtEth0/RP0/CPU0/0 192.168.122.22 Up Up default
```

Note
Use the **ip addr show** or **ip link show** commands to view all corresponding interfaces in Linux. The IOS XR interfaces that are `admin-down` state also reflects a `Down` state in the Linux kernel.

Step 3 Check the IP and MAC addresses of the interface that is in `Up` state. Here, interfaces `HundredGigE0/0/0/24` and `MgmtEth0/RP0/CPU0/0` are in the `Up` state.

Example:

```
Router#show interfaces HundredGigE0/0/0/24
...
HundredGigE0/0/0/24 is up, line protocol is up
Interface state transitions: 4
Hardware is HundredGigE0/0/0/24, address is 5246.e8a3.3754 (bia
5246.e8a3.3754)
Internet address is 10.1.1.1/24
MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
reliability 255/255, txload 0/255, rxload 0/255
Encapsulation ARPA,
Duplex unknown, 1000Mb/s, link type is force-up
output flow control is off, input flow control is off
loopback not set,
Last link flapped 01:03:50
ARP type ARPA, ARP timeout 04:00:00
Last input 00:38:45, output 00:38:45
Last clearing of "show interface" counters never
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
12 packets input, 1260 bytes, 0 total input drops
0 drops for unrecognized upper-level protocol
Received 2 broadcast packets, 0 multicast packets
0 runts, 0 giants, 0 throttles, 0 parity
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
12 packets output, 1224 bytes, 0 total output drops
Output 1 broadcast packets, 0 multicast packets
```

Step 4 Verify that the bash command runs in global VRF to view the network interfaces.

Example:

```
Router#bash -c ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:360 (360.0 B) TX bytes:0 (0.0 B)
Mg0_RP0_CPU0_0 Link encap:Ethernet HWaddr 54:00:00:00:bd:49
inet addr:192.168.122.22 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:3859 errors:0 dropped:0 overruns:0 frame:0
TX packets:1973 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:1000
RX bytes:2377782 (2.2 MiB) TX bytes:593602 (579.6 KiB)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:242 errors:0 dropped:0 overruns:0 frame:0
TX packets:242 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:12100 (11.8 KiB) TX bytes:12100 (11.8 KiB)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:60 (60.0 B)
```

The `to_xr` interface indicates access to the global VRF.

Step 5 Access the Linux shell.

Example:

```
Router#bash
[ios:~]$
```

Step 6 (Optional) View the IP routes used by the `to_xr` interfaces.

Example:

```
[ios:~]$ip route
default dev to_xr scope link metric 2048
6.1.0.0/16dev Mg0_RP0_CPU0_0 proto kernel scope link src 6.1.22.41
20.1.0.0/16dev Hu0_0_0_0 proto kernel scope link src 20.1.1.1
20.2.0.0/16dev Hu0_0_0_20 proto kernel scope link src 20.2.1.1
30.1.0.0/24dev BE500 proto kernel scope link src 30.1.0.1
172.17.0.0/16dev docker0 proto kernel scope link src 172.17.0.1linkdown
```

Note
You can also enter the global VRF directly after logging into IOS XR using the **run ip netns exec vrf-default bash** command.

Setting up Virtual IP Addresses

Feature Name	Release Information	Description
Virtual IP address in the Linux networking stack	Release 7.5.2	Virtual IP addresses allow a single IP address to connect to the current active RP after an RP switchover event. In addition, this functionality enables your network stack to support virtual IP addresses for third-party applications and IOS XR applications that use the Linux networking stack. The following commands are modified: • ipv4 virtual address • ipv6 virtual address • show linux networking interfaces address-only

Interfaces configured on IOS XR are programmed into the Linux kernel. These interfaces allow Linux applications to run as if they were running on a regular Linux system. This packet I/O capability ensures that off-the-shelf Linux applications can be run alongside IOS XR, allowing operators to use their existing tools and automate deployments with IOS XR.

The IP address on the Linux interfaces, MTU settings, MAC address are inherited from the corresponding settings of the IOS XR interface. Accessing the global VRF network namespace ensures that when you issue the **bash** command, the default or the global VRF in IOS XR is reflected in the kernel. This ensures default reachability based on the routing capabilities of IOS XR and the packet I/O infrastructure.

Virtual addresses can be configured to access a router from the management network such as gRPC using a single virtual IP address. On a device with two or more RPs, the virtual address refers to the management interface that is currently active. This functionality can be used across RP failover without the information of which RP is currently active. This is applicable to the Linux packet path.

Procedure

	Command or Action	Purpose
Step 1	You can use the following commands to verify the IP Address in the Linux networking stack:	• ipv4 virtual address • ipv6 virtual address • show linux networking interfaces address-only

Program Routes in Linux

The basic routes required to allow applications to send or receive traffic can be programmed into the kernel. The Linux network stack that is part of the kernel is used by normal Linux applications to send/receive packets. In an IOS XR stack, IOS XR acts as the network stack for the system. Therefore to allow the Linux network stack to connect into and use the IOS XR network stack, basic routes must be programmed into the Linux Kernel.

Procedure

Step 1 View the routes from the bash shell.

Example:

```
[ios:~]$ip route
default dev to_xr scope link src 10.1.1.10 metric 2048
10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
```

Step 2 Programme the routes in the kernel.

Two types of routes can be programmed in the kernel:

- **Default Route:** The default route sends traffic destined to unknown subnets out of the kernel using a special `to_xr` interface. This interface sends packets to IOS XR for routing using the routing state in XR Routing Information Base (RIB) or Forwarding Information Base (FIB). The `to_xr` interface does not have an associated IP address. In Linux, most applications expect the outgoing packets to use the IP address of the outgoing interface as the source IP address.

  With the `to_xr` interface, because there is no IP address, a source hint is required. The source hint can be changed to use the IP address another physical interface IP or loopback IP address. In the following example, the source hint is set to 10.1.1.10, which is the IP address of the Hu0_0_0_24 interface. To use the Management port IP address, change the source hint:

  ```
  Router#bash

  [ios:~]$ip route replace default dev to_xr scope link src 192.168.122.22 metric 2048

  [ios:~]$ip route
  default dev to_xr scope link src 192.168.122.22 metric 2048
  10.1.1.0/24 dev Hu0_0_0_24 proto kernel scope link src 10.1.1.10
  192.168.122.0/24 dev Mg0_RP0_CPU0_0 proto kernel scope link src 192.168.122.22
  ```

  With this updated source hint, any default traffic exiting the system uses the Management port IP address as the source IP address.

- **Local or Connected Routes:** The routes are associated with the subnet configured on interfaces. For example, the 10.1.1.0/24 network is associated with the `Hu0_0_0_24` interface, and the 192.168.122.0/24 subnet is associated with the `Mg0_RP0_CPU0` interface .

Configure VRFs in Linux

VRFs configured in IOS XR are automatically synchronized to the kernel. In the kernel, the VRFs appear as network namespaces (netns). For every globally-configured VRF, a Linux network namespace is created. With this capability it is possible to isolate Linux applications or processes into specific VRFs like an out-of-band management VRF and open-up sockets or send or receive traffic only on interfaces in that VRF.

Every VRF, when synchronized with the Linux kernel, is programmed as a network namespace with the same name as a VRF but with the string `vrf` prefixed to it. The default VRF in IOS XR has the name `default`. This name gets programmed as `vrf-default` in the Linux kernel.

The following example shows how to configure a custom VRF `blue`:

Procedure

Step 1 Identify the current network namespace or VRF.

Example:

```
[ios:~]$ip netns identify $$
vrf-default
global-vrf
```

Step 2 Configure a custom VRF `blue`.

Example:

```
Router#conf t

Router(config)#vrf blue
Router(config-vrf)#commit
```

Step 3 Verify that the VRF `blue` is configured in IOS XR.

Example:

```
Router#show run vrf
vrf blue
!
```

Step 4 Verify that the VRF `blue` is created in the kernel.

Example:

```
Router#bash

[ios:~]$ls -l /var/run/netns
total 0
-r--r--r--. 1 root root 0 Jul 30 04:17 default
-r--r--r--. 1 root root 0 Jul 30 04:17 global-vrf
-r--r--r--. 1 root root 0 Jul 30 04:17 tpnns
-r--r--r--. 1 root root 0 Aug 1 17:01 vrf-blue
-r--r--r--. 1 root root 0 Jul 30 04:17 vrf-default
-r--r--r--. 1 root root 0 Jul 30 04:17 xrnns
```

Step 5 Access VRF `blue` to launch and execute processes from the new network namespace.

Example:

```
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$
```

```
[ios:~]$ip netns identify $$
vrf-blue
[ios:~]$
```

Running an **ifconfig** command shows only the default `to-xr` interface because there is no IOS XR interface in this VRF.

```
[ios:~]$ifconfig
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
[ios:~]$
```

Step 6 Configure an interface in the VRF `blue` in IOS XR. This interface will be configured automatically in the network namespace `vrf-blue` in the kernel.

Example:

The following example shows how to configure HundredGigE 0/0/0/24 interface in `vrf-blue` from IOS XR:

```
Router#conf t
Router(config)#int HundredGigE 0/0/0/24
Router(config-if)#no ipv4 address
Router(config-if)#vrf blue
Router(config-if)#ipv4 address 10.1.1.10/24
Router(config-if)#commit
```

Step 7 Verify that the HundredGigE 0/0/0/24 interface is configured in the VRF `blue` in IOS XR.

Example:

```
Router#show run int HundredGigE 0/0/0/24
interface HundredGigE0/0/0/24
vrf blue
ipv4 address 10.1.1.10 255.255.255.0
!
```

Step 8 Verify that the interface is configured in the VRF `blue` in the kernel.

Example:

```
Router#bash
Thu Aug 1 17:09:39.314 UTC
[ios:~]$
[ios:~]$ip netns exec vrf-blue bash
[ios:~]$
[ios:~]$ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
```

```
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
[ios:~]$
```

Open Linux Sockets

The socket entries are programmed into the Local Packet Transport Services (LPTS) infrastructure that distributes the information through the line cards. Any packet received on a line card interface triggers an LPTS lookup to send the packet to the application opening the socket. Because the required interfaces and routes already appear in the kernel, the applications can open the sockets — TCP or UDP.

Procedure

Step 1 Verify that applications open up sockets.

Example:

```
Router#bash
[ios:~]$nc -l 0.0.0.0 -p 5000 &
[1] 1160
[ios:~]$
[ios:~]$netstat -nlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 0.0.0.0:5000 0.0.0.0:* LISTEN 1160/nc
tcp 0 0 0.0.0.0:57777 0.0.0.0:* LISTEN 14723/emsd
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 8875/ssh_server
tcp6 0 0 :::22 :::* LISTEN 8875/ssh_server
udp 0 0 0.0.0.0:68 0.0.0.0:* 13235/xr_dhcpcd
Active UNIX domain sockets (only servers)
Proto RefCnt Flags Type State I-Node PID/Program name Path
[ios:~]$exit
Logout
Router#
Router#show lpts pifib brief | i 5000
Thu Aug 1 17:16:00.938 UTC
IPv4 default TCP any 0/RP0/CPU0 any,5000 any
Router#
```

Step 2 Verify that the socket is open.

Example:

```
Router#show lpts pifib brief | i 5000
IPv4 default TCP any 0/RP0/CPU0 any,5000 any
```

Netcat starts listening on port 5000, which appears as an IPv4 TCP socket in the netstat output like a typical Linux kernel. This socket gets programmed to LPTS, creating a corresponding entry in the hardware to the lookup tcp port 5000. The incoming traffic is redirected to the kernel of the active RP where the netcat runs.

Send and Receive Traffic

Connect to the nc socket from an external server. For example, the nc socket was started in the `vrf-default` network namespace. So, connect over an interface that is in the same VRF.

```
[root@localhost ~]#nc -vz 192.168.122.22 5000
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Connected to 192.168.122.22:5000.
Ncat: 0 bytes sent, 0 bytes received in 0.01 seconds.
```

Manage IOS XR Interfaces through Linux

The Linux system contains a number of individual network namespaces. Each namespace contains a set of interfaces that map to a single interface in the XR control plane. These interfaces represent the exposed XR interfaces (eXI). By default, all interfaces in IOS XR are managed through the IOS XR configuration (CLI or YANG models), and the attributes of the interface (IP address, MTU, and state) are inherited from the corresponding configuration and the state of the interface in XR.

With the new Packet I/O functionality, it is possible to have an IOS XR interface completely managed by Linux. This also means that one or more of the interfaces can be configured to be managed by Linux, and standard automation tools can be used on Linux servers can be used to manage interfaces in IOS XR.

Note Secondary IPv4 addresses cannot be managed by Linux.

Configure an Interface to be Linux-Managed

This section shows how to configure an interface to be Linux-managed.

Procedure

Step 1 Check the available exposed-interfaces in the system.

Example:

```
Router(config)#linux networking exposed-interfaces interface ?

  Bundle-Ether     Aggregated Ethernet interface(s) | short name is BE
  FiftyGigE        FiftyGigabitEthernet/IEEE 802.3 interface(s) | short name is Fi
  FortyGigE        FortyGigabitEthernet/IEEE 802.3 interface(s) | short name is Fo
  FourHundredGigE  FourHundredGigabitEthernet/IEEE 802.3 interface(s) | short name is FH
  GigabitEthernet  GigabitEthernet/IEEE 802.3 interface(s) | short name is Gi
  HundredGigE      HundredGigabitEthernet/IEEE 802.3 interface(s) | short name is Hu
  Loopback         Loopback interface(s) | short name is Lo
  MgmtEth          Ethernet/IEEE 802.3 interface(s) | short name is Mg
  TenGigE          TenGigabitEthernet/IEEE 802.3 interface(s) | short name is Te
```

```
  TwentyFiveGigE   TwentyFiveGigabitEthernet/IEEE 802.3 interface(s) | short name is TF
  TwoHundredGigE   TwoHundredGigabitEthernet/IEEE 802.3 interface(s) | short name is TH
```

Step 2 Configure the interface to be managed by Linux.

Example:

The following example shows how to configure a HundredGigE interface to be managed by Linux:

```
Router#configure
Router(config)#linux networking exposed-interfaces interface HundredGigE 0/0/0/24 linux-managed
Router(config-exi-if)#commit
```

Step 3 View the interface details and the VRF.

Example:

The following example shows the information for HundredGigE interface:

```
Router#show run interface HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
mtu 4110
vrf blue
ipv4 mtu 4096
ipv4 address 10.1.1.10 255.255.255.0
ipv6 mtu 4096
ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
!
```

Step 4 Verify the configuration in XR.

Example:

The following example shows the configuration for HundredGigE interface:

```
Router#show running-config linux networking

linux networking
 exposed-interfaces
  interface HundredGigE0/0/0/24 linux-managed
  !
 !
!
```

Step 5 Verify the configuration from Linux.

Example:

The following example shows the configuration for HundredGigE interface:

```
Router#bash
Router:Aug 1 17:40:02.873 UTC: bash_cmd[67805]: %INFRA-INFRA_MSG-5-RUN_LOGIN : User vagrant logged
into shell from vty0

[ios:~]$ip netns exec vrf-blue bash

[ios:~]$ifconfig
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

```
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

[ios:~]$ifconfig -a
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Configure New IP address on the Interface in Linux

This section shows how to configure a new IP address on the Linux-managed interface.

Procedure

Step 1 Configure the IP address on the interface.

Example:

```
[ios:~]$ip addr add 10.1.1.10/24 dev Hu0_0_0_24
[ios:~]$Router:Aug 1 17:41:11.546 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'system'. Use 'show configuration commit changes 1000000021' to view the changes.
```

Step 2 Verify that the new IP address is configured.

Example:

```
[ios:~]$ifconfig Hu0_0_0_24
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
BROADCAST MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Configure Custom MTU Setting

This section shows how to bring up the interface and configure a custom MTU in a Linux-managed interface.

Procedure

Step 1 Configure the MTU setting.

Example:

```
[ios:~]$ifconfig Hu0_0_0_24 up

[ios:~]$Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN : Interface
HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:54.824 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN : Line protocol on
Interface HundredGigE0/0/0/24, changed state to Down
Router:Aug 1 17:41:56.448 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000022' to view the changes.
Router:Aug 1 17:41:56.471 UTC: ifmgr[266]: %PKT_INFRA-LINK-3-UPDOWN : Interface
HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:56.484 UTC: ifmgr[266]: %PKT_INFRA-LINEPROTO-5-UPDOWN : Line protocol on
Interface HundredGigE0/0/0/24, changed state to Up
Router:Aug 1 17:41:58.493 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration committed by
user 'system'. Use 'show configuration commit changes 1000000023' to view the changes.

[ios:~]$
[ios:~]$ ip link set dev Hu0_0_0_24 mtu 4096
[ios:~]$
[ios:~]$Router:Aug 1 17:42:46.830 UTC: xlncd[253]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
committed by user 'system'. Use 'show configuration commit changes 1000000024' to view the changes.
```

Step 2 Verify that the MTU setting has been updated in Linux.

Example:

```
[ios:~]$ifconfig
Hu0_0_0_24 Link encap:Ethernet HWaddr 78:e7:e8:d3:20:c0
inet addr:10.1.1.10 Bcast:0.0.0.0 Mask:255.255.255.0
inet6 addr: fe80::7ae7:e8ff:fed3:20c0/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:4096 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B) TX bytes:648 (648.0 B)
lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
to_xr Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1500 Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:500
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Step 3 Check the effect on the IOS XR configuration with the change in MTU setting on this interface.

Example:

```
Router#show running-config int HundredGigE0/0/0/24
interface HundredGigE0/0/0/24
 mtu 4110
  vrf blue
  ipv4 mtu 4096
  ipv4 address 10.1.1.10 255.255.255.0
  ipv6 mtu 4096
  ipv6 address fe80::7ae7:e8ff:fed3:20c0 link-local
  !
 !
!
Router#
Router#show ip int br | i HundredGigE0/0/0/24
HundredGigE0/0/0/24 10.1.1.10 Up Up blue
```

The output indicates that the interface acts as a regular Linux interface, and IOS XR configuration receives inputs from Linux.

Configure Traffic Protection for Linux Networking

Traffic protection provides a mechanism to configure Linux firewalls using IOS XR configuration. These rules can be used to restrict traffic to Linux applications. You can restrict traffic to Linux applications using native Linux firewalls or configuring IOS XR Linux traffic protection. It is not recommended to use both mechanisms at the same time. Any combination of remote address, local address and ingress interface can be specified as rules to either allow or deny traffic. However, at least one parameter must be specified for the traffic protection rule to be valid.

Note If traffic is received on a protocol or port combination that has no traffic protection rules configured, then all traffic is allowed by default.

This example explains how to configure a traffic protection rule on IOS XR to deny all traffic on port 999 except for traffic arriving on interface HundredGigE0/0/0/25.

Procedure

Step 1 Configure traffic protection rules.

Example:

```
Router(config)#linux networking vrf default address-family ipv4 protection protocol
tcp local-port 999 default-action deny permit hundredgigE0/0/0/25
Router(config)#commit
```

where —

- **address-family:** Configuration for a particular IPv4 or IPv6 address family.
- **protection:** Configure traffic protection for Linux networking.
- **protocol:** Select the supported protocol - TCP or UDP.

- **local-port:** L4 port number to specify traffic protection rules for Linux networking.
- **port number:** Port number ranges from 1 to 65535 or all ports.
- **default-action:** Default action to take for packets matching this traffic protection service.
- **deny:** Drop packets for this service.
- **permit:** Permit packets to reach Linux application for this service.

Step 2 Verify that the traffic protection rule is applied successfully.

Example:

```
Router(config)#show run linux networking
linux networking
 vrf default
  address-family ipv4
   protection
    protocol tcp local-port 999 default-action deny
     permit interface HundredGigE0/0/0/25
     !
    !
   !
 !
```

Communication Outside Cisco IOS XR

To communicate outside Cisco IOS XR, applications use the `fwdintf` interface address that maps to the `loopback0` interface or a configured Gigabit Ethernet interface address. For information on the various interfaces on IOS XR, see Application Hosting on the Cisco IOS XR Linux Shell.

To have an iPerf or Chef client on IOS XR communicate with its respective server outside IOS XR, you must configure an interface address as the source address on XR. The remote servers must configure this route address to reach the respective clients on IOS XR.

This section provides an example of configuring a Gigabit Ethernet interface address as the source address for external communication.

Using a Gigabit Ethernet Interface for External Communication

To configure a GigE interface on IOS XR for external communication, use these steps:

1. Configure a GigE interface.

```
RP/0/RP0/CPU0:ios(config)# interface GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 192.57.43.10 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# no shut
RP/0/RP0/CPU0:ios(config-if)# commit
Fri Oct 30 07:51:14.785 UTC
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# exit
```

2. Verify whether the configured interface is up and operational on IOS XR.

```
RP/0/RP0/CPU0:ios# show ipv4 interface brief
Fri Oct 30 07:51:48.996 UTC
```

```
Interface                      IP-Address      Status                Protocol
Loopback0                      1.1.1.1         Up                    Up
Loopback1                      8.8.8.8         Up                    Up
GigabitEthernet0/0/0/0         192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1         192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2         unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0            192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

3. Enter the Linux bash shell and verify if the configured interface is up and running.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash


[xr-vm_node0_RP0_CPU0:~]$ ifconfig
Gi0_0_0_0 Link encap:Ethernet  HWaddr 52:46:04:87:19:3c
          inet addr:192.164.168.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:4ff:fe87:193c/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Gi0_0_0_1 Link encap:Ethernet  HWaddr 52:46:2e:49:f6:ff
          inet addr:192.57.43.10  Mask:255.255.255.0
          inet6 addr: fe80::5046:2eff:fe49:f6ff/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:210 (210.0 B)

Mg0_RP0_CPU0_0 Link encap:Ethernet  HWaddr 52:46:12:7a:88:41
          inet addr:192.168.122.197  Mask:255.255.255.0
          inet6 addr: fe80::5046:12ff:fe7a:8841/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1514  Metric:1
          RX packets:3 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:294 (294.0 B)  TX bytes:504 (504.0 B)

fwd_ew    Link encap:Ethernet  HWaddr 00:00:00:00:00:0b
          inet6 addr: fe80::200:ff:fe00:b/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:4 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:392 (392.0 B)  TX bytes:532 (532.0 B)

fwdintf   Link encap:Ethernet  HWaddr 00:00:00:00:00:0a
          inet6 addr: fe80::200:ff:fe00:a/64 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:1482  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:1 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:672 (672.0 B)  TX bytes:672 (672.0 B)

lo:0      Link encap:Local Loopback
          inet addr:1.1.1.1  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
```

4. Exit the Linux bash shell and configure the GigE interface as the source address for external communication.

```
[xr-vm_node0_RP0_CPU0:~]$ exit

RP/0/RP0/CPU0:ios# config
Fri Oct 30 08:55:17.992 UTC
RP/0/RP0/CPU0:ios(config)# tpa address-family ipv4 update-source gigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config)# commit
Fri Oct 30 08:55:38.795 UTC
```

Note By default, the `fwdintf` interface maps to the `loopback0` interface for external communication. This is similar to binding a routing process or router ID to the `loopback0` interface. When you use the `tpa address-family ipv4 update-source` command to bind the `fwdintf` interface to a Gigabit Ethernet interface, network connectivity can be affected if the interface goes down.

5. Enter the Linux bash shell and verify whether the GigE interface address is used by the `fwdintf` interface for external communication.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash

[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

External communication is successfully enabled on IOS XR.

East-West Communication for Third-Party Applications

East-West communication on IOS XR is a mechanism by which applications hosted in containers interact with native XR applications (hosted in the XR control plane).

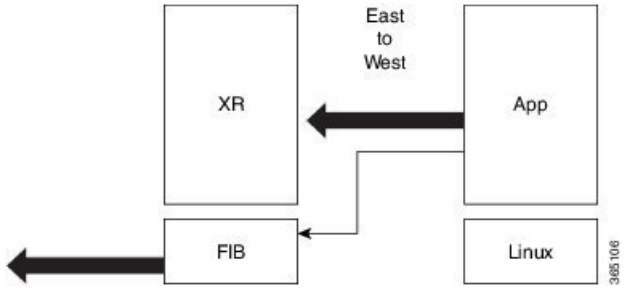
The following figure illustrates how a third-party application hosted on IOS XR interacts with the XR Control Plane.

The application sends data to the Forwarding Information Base (FIB) of IOS XR. The application is hosted in the east portion of IOS XR, while the XR control plane is located in the west region. Therefore, this form of communication between a third-party application and the XR control plane is termed as East-West (E-W) communication.

Third-party applications such as Chef Client and Puppet Agent use this mode of communication to configure and manage containers, packages, and applications on IOS XR. In the future, this support could be extended to IOS XR, configured and managed by such third-party applications.

Note East-West communication is not supported on IOS XR from software release 7.9.1.

Figure 1: East-West Communication on IOS XR



For a third-party application to communicate with IOS XR, the Loopback1 interface must be configured. This is explained in the following procedure.

1. Configure the Loopback1 interface on IOS XR.

```
RP/0/RP0/CPU0:ios(config)# interface Loopback1
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 8.8.8.8/32
RP/0/RP0/CPU0:ios(config-if)# no shut
RP/0/RP0/CPU0:ios(config-if)# commit
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)#
```

2. Verify the creation of the Loopback1 interface.

```
RP/0/RP0/CPU0:ios# show ipv4 interface brief
Thu Nov 12 10:01:00.874 UTC

Interface                  IP-Address      Status                Protocol
Loopback0                  1.1.1.1         Up                    Up
Loopback1                  8.8.8.8         Up                    Up
GigabitEthernet0/0/0/0     192.164.168.10  Up                    Up
GigabitEthernet0/0/0/1     192.57.43.10    Up                    Up
GigabitEthernet0/0/0/2     unassigned      Shutdown              Down
MgmtEth0/RP0/CPU0/0        192.168.122.197 Up                    Up
RP/0/RP0/CPU0:ios#
```

3. Enter the third-party network namespace or global VRF depending on the version of IOS XR version you are using for your network.

```
/* If you are using Cisco IOS XR Version 6.0.0, run the following command */
RP/0/RP0/CPU0:ios# run ip netns exec tpnns bash

/* If you are using Cisco IOS XR Version 6.0.2, run the following command */
RP/0/RP0/CPU0:ios# bash
```

4. Verify whether the Loopback1 interface address has been mapped to the E-W interface.

```
[xr-vm_node0_RP0_CPU0:~]$ ip route
default dev fwdintf  scope link  src 192.57.43.10
8.8.8.8 dev fwd_ew  scope link
192.168.122.0/24 dev Mg0_RP0_CPU0_0  proto kernel  scope link  src 192.168.122.197
[xr-vm_node0_RP0_CPU0:~]$
```

Telemetry Data Export from Line Card Port via User Defined VRF

Table 1: Feature History Table

Feature Name	Release Information	Description
Telemetry Data Export from Line Card Port via User Defined VRF	Release 24.1.1	You can now ensure that the collection and transmission of telemetry data does not interfere with your primary network operations, thus enhancing network isolation and security by reducing the risk of the data being tampered with or intercepted. This is achieved by exporting telemetry data from the router to your specified destination address through line card data ports, using either user-defined or default VRFs.

Starting from Cisco IOS XR 24.1.1 software release, we ensure that the collection and transmission of telemetry data do not interfere with your primary network operations, thus enhancing network isolation and security by reducing the risk of the data being tampered with or intercepted. This is achieved by exporting telemetry data from user-defined VRFs to a specified destination address through data ports of line cards. Using this functionality, you can export telemetry data through user-defined or default VRFs.

Prior to this release, telemetry data export was supported only through the default VRF. By leveraging data ports of line cards and user-defined VRFs, this release expands the scope of telemetry data export configurations on the Cisco ASR 9000 platform.

Configuring Telemetry Data Export from Line Card Port

This section provides you with the configuration steps for enabling a line card port for telemetry data forwarding using a user-defined VRF.

1. This example shows how to configure an interface for telemetry data export.

```
Router#config
Router(config)#interface TenGigE0/2/0/3
Router(config-if)#ipv4 address 111.1.1.1/24
Router(config-if)#vrf vrf_telemetry
```

2. This example shows how to configure gRPC on a user-defined port for telemetry data export.

```
Router(config)#grpc
Router(config-grpc)#port 57400
Router(config-grpc)#vrf vrf_telemetry
```

```
Router(config-grpc)#!
Router#grpc no-tls
```

Verification

This example shows how to verify VRF forwarding through user defined port.

```
Router# openconfig_gnmi_client get --path
"openconfig-interfaces:interfaces/interface[name=MgmtEth0/RSP1/CPU0/0]/state/mtu" --format
json -t OPERATIONAL --insecure -u cafyauto -p cisco123 -e json_ietf --timeout 120s -a
111.1.1.1:57400
[
{
"source": "111.1.1.1:57400",
"timestamp": 1702969098985896297,
"time": "2023-12-19T06:58:18.985896297Z",
"updates": [
{
"Path": "openconfig:interfaces/interface[name=MgmtEth0/RSP1/CPU0/0]/state/mtu",
"values": {
"interfaces/interface/state/mtu": 1514
}
}
]
}
]
```

The "source" field shows the source for VRF forwarding, and the "value" field indicates the retrieved VRF MTU value of 1514 bytes.

MPLS Explicit Null Label for IPv6 Data Packets

Table 2: Feature History Table

Feature Name	Release Information	Description
MPLS Explicit Null Label for IPv6 Data packets	Release 24.1.1	You now get a comprehensive insight into telemetry data for IPv6 gRPC sessions for Linux Third Party Applications (TPAs) within your network, providing data for swift identification and resolution of network issues as they arise. This is achieved by supporting IPv6 explicit null labels, which allows MPLS-labeled packets to be forwarded to Linux TPAs without removing their labels.

Starting with the Cisco IOS XR 24.1.1 software release, you can now retain MPLS labels for telemetry streaming of gRPC sessions over IPv6 traffic and gain visibility over MPLS labeled packets in your network. This empowers you to handle telemetry gRPC streams in-band, where IPv6 traffic is carried over RSVP-TE IPv4 tunnels. Previously, explicit null was only supported on IPv4 traffic. The MPLS labeled packets destined for Linux TPA are processed and forwarded to their respective TPAs, ensuring that their MPLS labels are

retained and received at the egress router without them being removed. This is achieved by using the **explicit-null** command.

Note The null label does not use the Border Gateway Protocol (BGP) to transport labels.

Configuration to Enable Explicit Null

This example shows how to configure explicit null label.

```
Router(config)# mpls ldp
Router(config-ldp)# address-family ipv6
Router(config-ldp-af)# label local advertise explict-null
```

Verification

This example shows how to verify explicit null label configuration.

```
Router(config)# show cef ipv6 58:1:1:1:: detail
58:1:1:1::/64, version 15294, internal 0x1000001 0x20 (ptr 0xa2ea9c68) [1], 0x600
(0xbeb55ee8), 0xa28 (0xa4610ab0)
 Updated Dec 19 06:45:20.809
 Prefix Len 64, traffic index 0, precedence n/a, priority 2, encap-id 0x100c700000001
  gateway array (0xc020ca58) reference count 2, flags 0x28, source rib (7), 0 backups
                [3 type 1 flags 0x8401 (0xc1004ab8) ext 0x0 (0x0)]
  LW-LDI[type=1, refc=1, ptr=0xbeb55ee8, sh-ldi=0xc1004ab8]
  gateway array update type-time 1 Dec 19 06:45:20.809
 LDI Update time Dec 19 06:45:20.809
 LW-LDI-TS Dec 19 06:45:20.809
 Accounting: Disabled
   via ::/128, named_TE_R2_R1_, 5 dependencies, weight 0, class 0 [flags 0x0]
    path-idx 0 NHID 0x0 [0xc1ee20a0 0x0]
    next hop VRF - 'default', table - 0xe0000000
    next hop ::/128
    local adjacency
     labels imposed {ExpNullv6}

    Load distribution: 0 (refcount 3)

    Hash  OK  Interface                 Address
    0     Y   named_TE_R2_R1_           point2point

[
{
"source": "[58:1:1:1::1]:57400",
"timestamp": 1702969041524464897,
"time": "2023-12-19T06:57:21.524464897Z",
"updates": [
{
"Path": "openconfig:interfaces/interface[name=MgmtEth0/RSP1/CPU0/0]/state/mtu",
"values": {
"interfaces/interface/state/mtu": 1514
}
}
]
}
]
```

The label imposed field shows explicit null enabled.