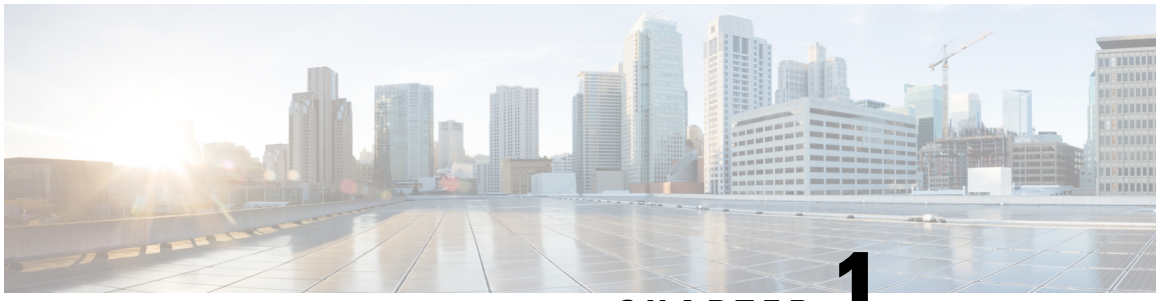
Application Hosting Configuration Guide for Cisco NCS 540 Series Routers

First Published: 2023-12-01

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS



CHAPTER 1

Getting Started with Application Hosting

This section introduces application hosting and the Linux environment used for hosting applications on the Cisco IOS XR Operating System.

Cisco NCS 540 routers supports docker-based application hosting only.

The following NCS 540 routers do not support application hosting:

- N540X-4Z14G2Q-D/A
- N540X-8Z16G-SYS-D/A
- N540X-6Z18G-SYS-D/A
- N540-6Z18G-SYS-D/A
- N540-6Z14S-SYS-D

Need for Application Hosting

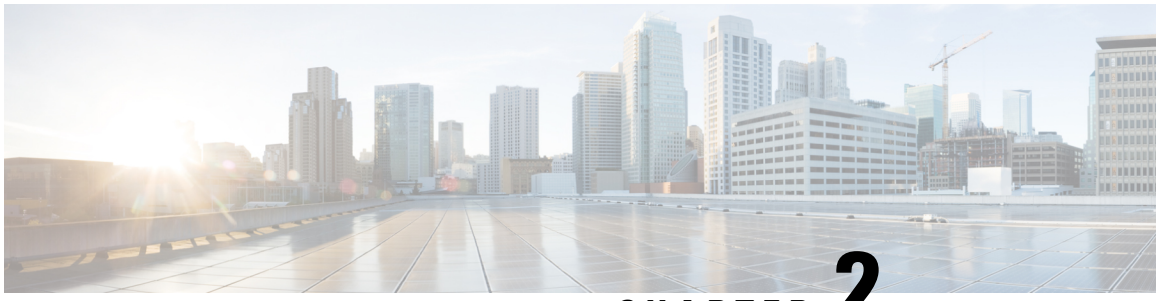
Over the last decade, there has been a need for a network operating system that supports operational agility and efficiency through seamless integration with existing tool chains. Service providers have been looking for shorter product cycles, agile workflows, and modular software delivery; all of these can be automated efficiently. It does that by providing an environment that simplifies the integration of applications, configuration management tools, and industry-standard zero touch provisioning mechanisms. The IOS XR matches the DevOps style workflows for service providers, and it has an open internal data storage system that can be used to automate the configuration and operation of the device hosting an application.

While we are rapidly moving to virtual environments, there is an increasing need to build applications that are reusable, portable, and scalable. Application hosting gives administrators a platform for leveraging their own tools and utilities. Cisco NCS 540 routers support third-party off-the-shelf applications. An application hosted on a network device can serve a variety of purposes. This ranges from automation, configuration management monitoring, and integration with existing tool chains.

Before an application can be hosted on a device, the following requirements must be met:

- Suitable build environment to build your application
- A mechanism to interact with the device and the network outside the device

When network devices are managed by configuration management applications, network administrators are freed of the task of focusing only on the CLI. Because of the abstraction provided by the application, while the application does its job, administrators can now focus on the design, and other higher level tasks.



CHAPTER 2

Hosting Applications on IOS XR

This section explains the different kinds of application hosting, and demonstrates how a simple application can be hosted natively or in a third-party container on IOS XR.

Application Hosting Using Docker Containers

Application hosting on IOS XR supports docker containers. You can create your own container on IOS XR using docker, and host applications within the container. The applications can be developed using any Linux distribution. This is well suited for applications that use system libraries that are different from that provided by the IOS XR root file system. Cisco NCS 540 supports only docker-based application hosting.

Docker-Based Container Application Hosting

This section introduces the concept of container application hosting and describes its workflow.

Container application hosting makes it possible for applications to be hosted in their own environment and process space (namespace) within a Linux container on Cisco IOS XR. The application developer has complete control over the application development environment, and can use a Linux distribution of choice. The applications are isolated from the IOS XR control plane processes; yet, they can connect to networks outside XR through the XR GigE interfaces. The applications can also easily access local file systems on IOS XR.

Using Docker for Hosting Applications on Cisco IOS XR

Docker is a container used for hosting applications on Cisco IOS XR. Docker provides isolation for application processes from the underlying host processes on XR by using Linux network namespaces.

Need for Docker on Cisco IOS XR

Docker is becoming the industry-preferred packaging model for applications in the virtualization space. Docker provides the foundation for automating application life cycle management.

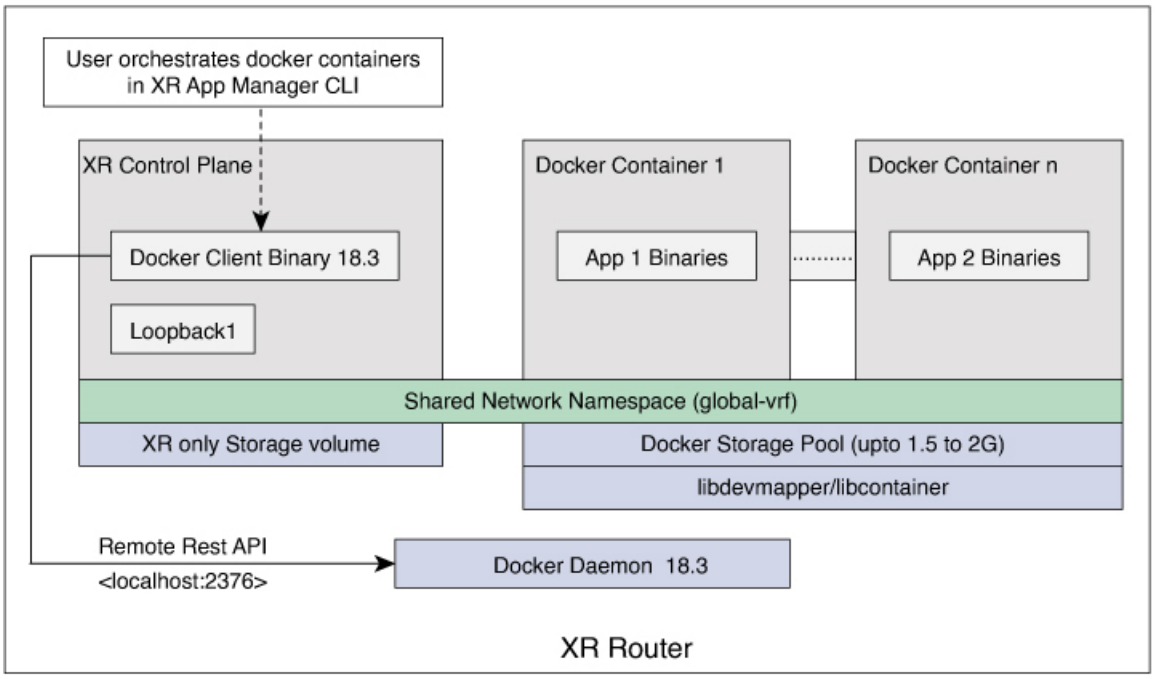
Docker follows a layered approach that consists of a base image at the bottom that supports layers of applications on top. The base images are available publicly in a repository, depending on the type of application you want to install on top. You can manipulate docker images by using the docker index and registry.

Docker provides a git-like workflow for developing container applications and supports the "thin update" mechanism, where only the difference in source code is updated, leading to faster upgrades. Docker also provides the "thin download" mechanism, where newer applications are downloaded faster because of the sharing of common base docker layers between multiple docker containers. The sharing of docker layers between multiple docker containers leads to lower footprint for docker containers on XR.

Docker Architecture on Cisco IOS XR

The following figure illustrates the docker architecture on IOS XR.

Figure 1: Docker Workflow for Updating Applications

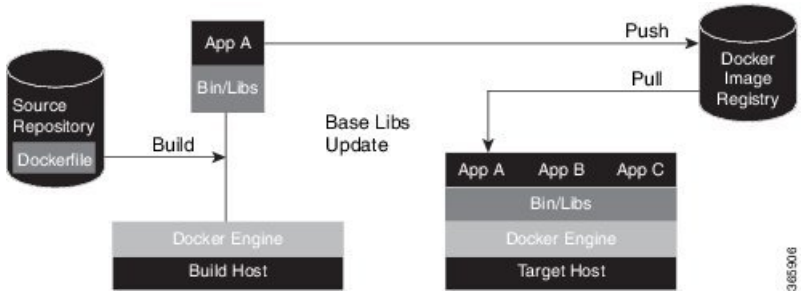


The application binaries for the applications to be hosted are installed inside the docker container.

Hosting Applications in Docker Containers

The following figure illustrates the workflow for hosting applications in Docker containers on IOS XR.

Figure 2: Docker Workflow for Application Hosting



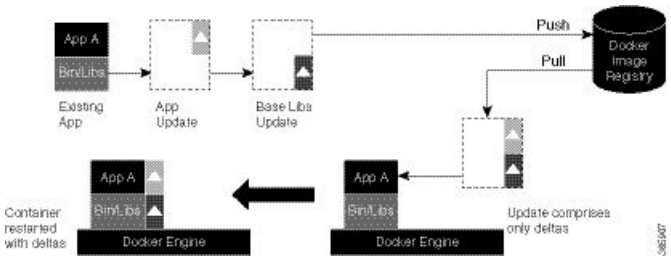
1. The docker file in the source repository is used to build the application binary file on your (docker engine build) host machine.

2. The application binary file is pushed into the docker image registry.
3. The application binary file is pulled from the docker image registry and copied to the docker container on XR (docker engine target host).
4. The application is built and hosted in the docker container on XR.

Updating Applications in Docker Containers

The following figure illustrates the workflow for updating applications hosted in docker containers.

Figure 3: Docker Workflow for Updating Applications



1. The application update is generated as a base libs update file (delta update file) and pushed to the docker image registry.
2. The delta update file (containing only the difference in application code) is pulled from the docker image registry and copied to the docker containers on XR (docker engine target host).
3. The docker containers are restarted with the delta update file.

Hosting of TPA Using Application Manager

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
On-Demand Docker Daemon Service	Release 7.5.1	From this release onwards, the Docker daemon service starts on a router only if you configure a third-party hosting application using the **appmgr** command. Such an on-demand service optimizes operating system resources such as CPU, memory, and power. In earlier releases, the Docker daemon service automatically started during the router boot up.

In previous releases, the applications were hosted and controlled by the Docker commands. These Docker commands were executed in the bash shell of the Kernel that also hosted the Cisco IOS XR software. With the introduction of Application Manager, it is now possible to manage third-party application hosting and their functioning through Cisco IOS XR CLIs. With this feature, all the activated third party applications can

restart automatically after a router reload or an RP switchover. This automatic restart of the applications ensure seamless functioning of the hosted applications.

Supported Commands on Application Manager

For every application manager command or configuration executed, the Application Manager performs the requested action by interfacing with the Docker daemon through the Docker socket.

The following table lists the Docker container functionalities, the generic Docker commands that were used in the previous releases, and its equivalent application manager commands that can now be used:

Functionality	Generic Docker Commands	Application Manager Commands
Install the application RPM	NA	`Router#appmgr package install rpm image_name-0.1.0-XR_7.3.1.x86_64.rpm`

Configure and activate the application	• Load image - `[xr-vm_node0_RP0_CPU0:~]$docker load -i /tmp/image_name.tar`
	• Verify the image on the router - `xr-vm_node0_RP0_CPU0:~]$docker images ls`
	• Create container over the image - `[xr-vm_node0_RP0_CPU0:~]$docker create image_name`
	• Start container - `[xr-vm_node0_RP0_CPU0:~]$docker start my_container_id`
		`Router#config`
		`Router(config)#appmgr`
		`Router(config-appmgr)#application app_name`
		`Router(config-application)#activate type docker source image_name docker-run-opts "--net=host" docker-run-cmd "iperf3 -s -d"`
		`Router(config-application)#commit`

View the list, statistics, logs, and details of the application container	• List images -`[xr-vm_node0_RP0_CPU0:~]$docker images ls`
	• List containers - `[xr-vm_node0_RP0_CPU0:~]$docker ps`
	• Statistics -`[xr-vm_node0_RP0_CPU0:~]$docker stats`
	• Logs -`[xr-vm_node0_RP0_CPU0:~]$docker logs`
		`Router#show appmgr source-table`
		`Router#show appmgr application name app_name info summary`
		`Router#show appmgr application name app_name info detail`
		`Router#show appmgr application name app_name stats`
		`Router#show appmgr application-table`
		`Router#show appmgr application name app_name logs`

Functionality	Generic Docker Commands	Application Manager Commands
Run a new command inside a running container	• Execute - `[xr-vm_node0_RP0_CPU0:~]$docker exec -it my_container_id`	`Router#appmgr application exec name app_name docker-exec-cmd`
Stop the application container	• Stop container - `[xr-vm_node0_RP0_CPU0:~]$docker stop my_container_id`	`Router#appmgr application stop name app_name`
Kill the application container	• Kill container - `[xr-vm_node0_RP0_CPU0:~]$docker kill my_container_id`	`Router#appmgr application kill name app_name`
Start the application container	• Start container - `[xr-vm_node0_RP0_CPU0:~]$docker start my_container_id`	`Router#appmgr application start name app_name`

Deactivate the application	• Stop container - `[xr-vm_node0_RP0_CPU0:~]$docker stop my_container_id`
	• Remove container - `[xr-vm_node0_RP0_CPU0:~]$docker rm my_container_id`
	• Remove image - `[xr-vm_node0_RP0_CPU0:~]$docker rmi image_name`
		`Router#configure`
		`Router(config)#no appmgr application app_name`
		`Router(config)#commit`

Uninstall the application image/RPM	• Uninstall image - `[xr-vm_node0_RP0_CPU0:~]$docker app uninstall image_name`	`Router#appmgr package uninstall package image_name-0.1.0-XR_7.3.1.x86_64`

Note The usage of the application manager commands are explained in the "Hosting iPerf in Docker Containers to Monitor Network Performance using Application Manager" section.

Configuring a Docker with Multiple VRFs

This section describes how you can configure a Docker with multiple VRFs on Cisco IOS XR. For information on configuring multiple VRFs, see Configuring Multiple VRFs for Application Hosting topic.

Configuration

Use the following steps to create and deploy a multi-VRF Docker on XR.

1. Create a multi-VRF Docker with NET_ADMIN and SYS_ADMIN privileges.

   In the following example, a Docker container containing three VRFs (yellow, blue, and green) is launched. The example assumes that a previous "multivrfimage" docker image was installed using the `appmgr package install` command.

```
Router# appmgr application multivrfcontainer activate type docker source multivrfimage
docker-run-opts "-td --net=host --name multivrfcontainer1
-v /var/run/netns/yellow:/var/run/netns/yellow
-v /var/run/netns/blue:/var/run/netns/blue
-v /var/run/netns/green:/var/run/netns/green
--cap-add NET_ADMIN --cap-add SYS_ADMIN"
```

Note

- Mounting the entire content of `/var/run/netns` from host to Docker is not recommended, because it mounts the content of `netns` corresponding to XR and the system admin plane into the Docker.
- You should not delete a VRF from Cisco IOS XR when it is used in a Docker. If one or more VRFs are deleted from XR, the multi-VRF Docker cannot be launched.

2. Verify if the multi-VRF Docker has been successfully loaded.

```
Router# show appmgr application-table
Name                Type     Config     State       Status
---- ------ ------------ ------------------------------------------------------
multivrfcontainer  Docker   Activated    Up    About a minute
```

3. Connect to the multi-VRF Docker container by executing the following command.

```
Router# appmgr application exec name multivrfcontainer1 docker-exec-cmd /bin/bash/
```

   By default, the Docker is loaded in global-vrf namespace on Cisco IOS XR.

4. Verify if the multiple VRFs are accessible from the Docker.

```
root@host:/# ifconfig
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: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)

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:1500  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: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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

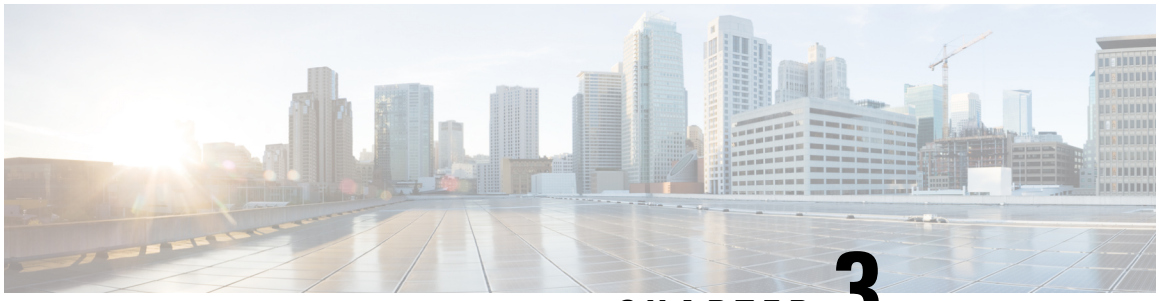
```
root@host:/# ip netns list
yellow
green
blue

root@host:/# /sbin/ip netns exec green bash
root@host:/# ifconfig -a
lo        Link encap:Local Loopback
          LOOPBACK  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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@host:/# ifconfig lo up
root@host:/# ifconfig lo 127.0.0.2/32
root@host:/# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.2  Mask:0.0.0.0
          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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)


[host:/misc/app_host]$ ip netns exec green bash
[host:/misc/app_host]$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.2  Mask:0.0.0.0
          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:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

You have successfully launched a multi-VRF Docker on Cisco IOS XR.



CHAPTER 3

Setup the Linux Network for Application Hosting

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.

Note This section is applicable only for the NCS 540 L platform, which runs the LNT Linux packet I/O for third-party applications. The configuration commands are run in the **linux-networking** submode.

Setting up Virtual IP Addresses

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.

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.

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.

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.

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 Program the routes in the Linux 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 .

Program the Source Hint for Linux Default Routes from XR CLI

You can configure the source hint for Linux default routes by using the IOS XR CLI.

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 Configure a source hint for the default route.

```
linux networking vrf <vrf-name> address-family <ipv4/ipv6> source-hint
{ default-route { <interface> | active-management }
| management-route <interface>}
```

By specifying the source hint, XR specifies the source address that Linux should use while sending traffic to the XR 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`:

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.

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.

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.

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.

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.

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
     !
    !
   !
 !
```

Synchronize Statistics Between IOS XR and Linux

This example shows how the bundle-ether interface packet statistics are synchronized between IOS XR and Linux. The packet and byte counters maintained by Linux for IOS XR interfaces display only the traffic

sourced in Linux. You can configure to periodically synchronize these counters with the IOS XR statistics for the interfaces.

Step 1 Configure the statistics synchronization including the direction and synchronization interval.

Example:

The following example shows statistics synchronization in global configuration:

```
Router(config)#linux networking statistics-synchronization from-xr
every 30s
```

Example:

The following example shows statistics synchronization in exposed-interface configuration:

```
Router(config)#linux networking exposed-interfaces interface
bundle-ether 1 statistics-synchronization from-xr every 10s
```

where —

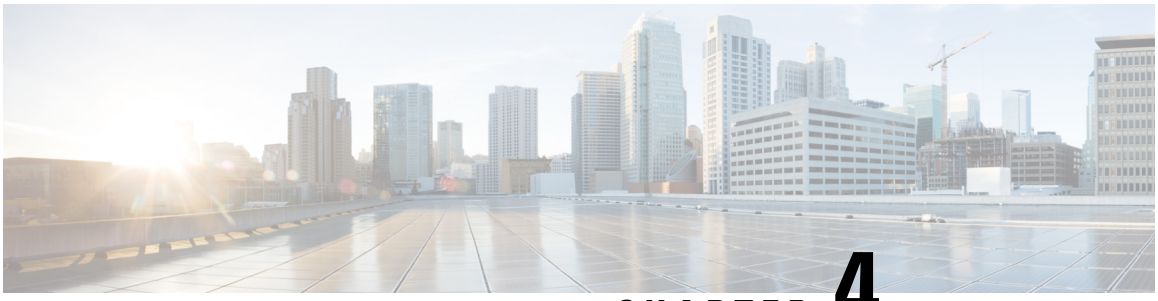
- **from-xr:** The direction indicating that the interface packet statistics will be pushed from IOS XR to the Linux kernel.
- **every:** Shows the frequency at which to synchronize statistics. The intervals supported for global configuration are 30s and 60s. The intervals supported for exposed interfaces are 5s, 10s, 30s or 60s. The interval `s` is in seconds.

Step 2 Verify that the statistics synchronization is applied successfully on IOS XR.

Example:

```
Router#show run linux networking
linux networking
 vrf default
  address-family ipv4
   protection
   protocol tcp local-port all default-action deny
    permit interface bundle-ether 1
    !
   !
  !
 !
exposed-interfaces
 interface bundle-ether 1 linux-managed
  statistics-synchronization from-xr every 10s
  !
 !
!
```

For troubleshooting purposes, use the **show tech-support linux networking** command to display debugging information.



CHAPTER 4

Use Cases: Application Hosting

This chapter describes use cases for running applications on IOS XR.

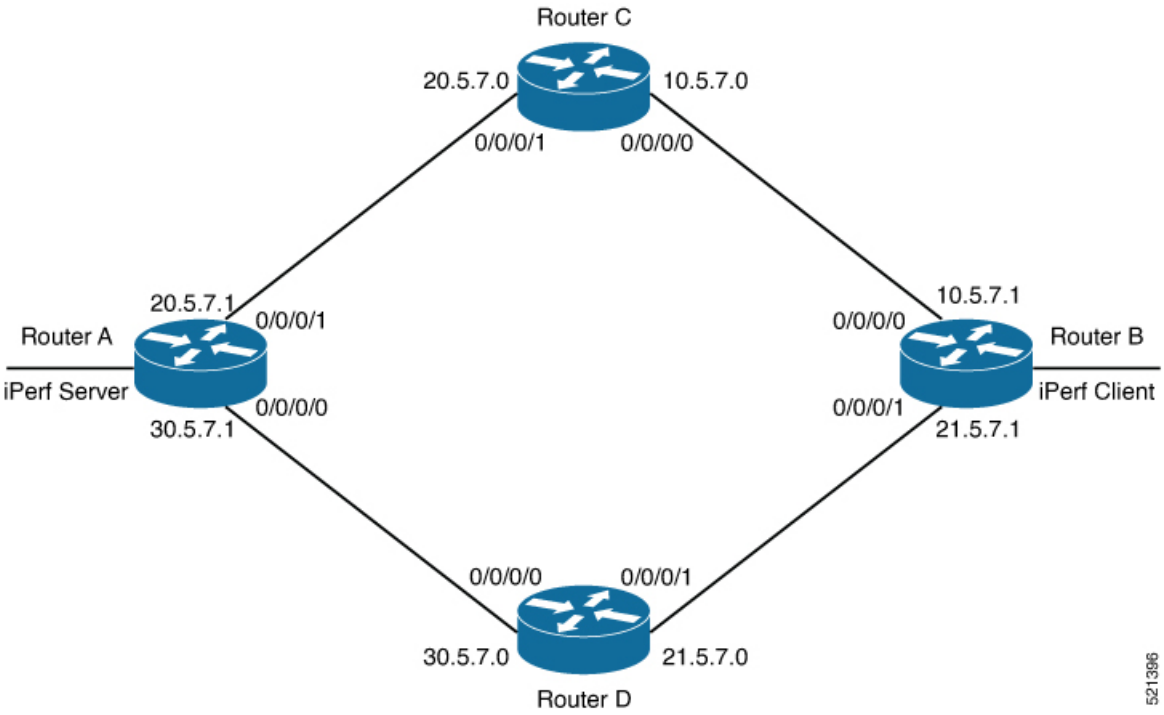
Hosting iPerf in Docker Containers to Measure Network Performance using Application Manager

Measuring the network performance is important to test the efficiency of the network. Network throughput, bandwidth, latency, and packet loss are some of the parameters used to measure the network performance. iPerf is a commonly used application for measuring network performance. The iPerf application is hosted on systems at both ends of the connection that is measured. One system is used as the server, and the other system is used as the client. At least one system must be a Cisco IOS XR router, the other system can be any other external entity like a controller or another router.

This use case illustrates the procedure for hosting the iPerf application in docker containers on two Cisco IOS XR routers, Router A and Router B to measure network performance. Router A hosts the iPerf server and Router B hosts the iPerf client.

In this usecase, we demonstrate the example of testing network bandwidth when a route update takes place. Router A hosts the iPerf Server and Router B hosts the iPerf Client. Router C and Router D are intermediate routers that allow traffic flow from Router A to Router B and vice-versa.

Figure 4: Hosting iPerf Application in Cisco IOS XR Routers



Verify Connection between the iPerf Server and iPerf Client Applications

Verify whether the connection is established between iPerf server and iPerf clients by executing the **bash netstat -anput** command on Router A. When the iPerf client is up and running, the entry in the **State** field displays "ESTABLISHED".

```
Router#bash netstat -anput
Thu Dec  3 10:00:33.535 UTC
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:646             0.0.0.0:*               LISTEN      8585/mpls_ldp
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      8567/ssh_server
tcp        0      0 0.0.0.0:830             0.0.0.0:*               LISTEN      8567/ssh_server
tcp6       0      0 :::5201                 :::*                    LISTEN      20829/iperf3
tcp6       0      0 :::22                   :::*                    LISTEN      8567/ssh_server
tcp6       0      0 :::830                  :::*                    LISTEN      8567/ssh_server
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65322         ESTABLISHED 20829/iperf3
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65302         ESTABLISHED 20829/iperf3
udp        0      0 0.0.0.0:646             0.0.0.0:*                           8585/mpls_ldp
udp        0      0 0.0.0.0:3232            0.0.0.0:*                           6833/pim
udp        0      0 0.0.0.0:3503            0.0.0.0:*                           10762/lspv_server
udp        0      0 0.0.0.0:68              0.0.0.0:*                           10704/xr_dhcpcd
udp        0      0 0.0.0.0:496             0.0.0.0:*                           6833/pim
udp6       0      0 :::3503                 :::*                                10762/lspv_server
```

Install the iPerf Server Application

Step 1 Install the iPerf application RPM on Router A. Only the RPM file format is supported.

```
Router#appmgr package install rpm /misc/disk1/iperf-0.1.0-XR_7.3.1.x86_64.rpm

Router#show appmgr source-table
Thu Dec  3 09:57:40.808 UTC
Name            File
--------------- ------------------------------------------------------------
iperf    iperf.tar.gz
Router#
```

Step 2 Configure the application to run as iPerf server.

```
Router#config
Thu Dec  3 09:57:54.034 UTC
Router(config)#appmgr
Router(config-appmgr)#application iperf-server-app
Router(config-application)#activate type docker source iperf docker-run-opts "--net=host" docker-run-cmd
 "iperf3 -s -d"
Router(config-application)#commit
Thu Dec  3 09:57:54.398 UTC
```

Step 3 Verify the basic details (application name and state) about the activated iPerf server application.

```
Router#show appmgr application-table
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   Up 2 seconds
Router#
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-server-app info summary
Thu Dec  3 09:58:15.569 UTC
Application: iperf-server-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Container ID: 0118f9006cde2787e9809eb7c62ad8b552925b559a689c7aaa80f80d7ce43c02
    Image: alpine1:latest
    Command: "iperf3 -s -d"
    Status: Up 7 seconds
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-server-app info detail
Thu Dec  3 09:58:26.401 UTC
Application: iperf-server-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Docker Information:
        Container ID: 0118f9006cde2787e9809eb7c62ad8b552925b559a689c7aaa80f80d7ce43c02
        Container name: iperf-server-app
        Labels:
        Image: alpine1:latest
        Command: "iperf3 -s -d"
        Created at: 2020-12-03 09:58:08 +0000 UTC
        Running for: 18 seconds ago
        Status: Up 18 seconds
        Size: 0B
        Ports:
        Mounts:
        Networks: host
```

```
       LocalVolumes: 0
Router#show appmgr application name iperf-server-app stats
Thu Dec  3 09:58:39.594 UTC
Application Stats: iperf-server-app
   CPU Percentage: 0.00%
   Memory Usage: 624KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 1
Router#
```

Step 4 Verify if the iPerf server is listening on the default port (5201) by using the netstat command inside the container.

The appmgr application exec name *app_name* docker-exec-cmd command can be used to execute any commands inside the container.

```
Router#appmgr application exec name iperf-server-app docker-exec-cmd name netstat -lnput
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.11:46727        0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:5201          0.0.0.0:*               LISTEN      -
udp        0      0 127.0.0.11:39552        0.0.0.0:*
Router#
```

Install the iPerf Client Application

Step 1 Install the iPerf application RPM on Router B.

```
Router#appmgr package install rpm /misc/disk1/iperf-0.1.0-XR_7.3.1.x86_64.rpm
Router#show appmgr source-table
Thu Dec  3 09:57:40.808 UTC
Name            File
--------------- ------------------------------------------------------------
iperf      iperf.tar.gz
Router#
```

Step 2 Configure the application to run as iPerf client with a timeout (600s in this case).

```
Router#config
Thu Dec  3 09:57:54.034 UTC
Router(config)#appmgr
Router(config-appmgr)#application iperf-client-app
Router(config-application)#activate type docker source iperf docker-run-opts "--net=host" docker-run-cmd
 "iperf3 -c 30.5.7.1 -t 600"
Router(config-application)#commit
Thu Dec  3 09:57:54.398 UTC
```

Note Hosting the iPerf client application on Router B by providing the iPerf server physical interface IP address (30.5.7.1) establishes communication between Router B and Router A.

Step 3 Verify the basic details (application name and state) about the activated iPerf client application.

```
Router#show appmgr application-table
Thu Dec  3 09:59:47.628 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-client-app     Docker    Activated   Up 2 seconds
```

```
Router#
Thu Dec  3 09:57:54.398 UTC
Router#show appmgr application name iperf-client-app info summary
Thu Dec  3 09:59:54.534 UTC
Application: iperf-client-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Container ID: 40e1730a97666b2b44c8c9313b94b0138925c9198ae63244ff3bd386132d9c9c
    Image: alpine1:latest
    Command: "iperf3 -c 30.5.7.1 -t 600"
    Status: Up 9 seconds
Router#show appmgr application name iperf-client-app info detail
Application: iperf-client-app
    Type: Docker
    Source: iperf
    Config State: Activated
    Docker Information:
        Container ID: 40e1730a97666b2b44c8c9313b94b0138925c9198ae63244ff3bd386132d9c9c
        Container name: iperf-client-app
        Labels:
        Image: alpine1:latest
        Command: "iperf3 -c 30.5.7.1 -t 600"
        Created at: 2020-12-03 09:59:45 +0000 UTC
        Running for: 20 seconds ago
        Status: Up 20 seconds
        Size: 0B
        Ports:
        Mounts:
        Networks: host
        LocalVolumes: 0
Router#show appmgr application name iperf-client-app stats
Thu Dec  3 10:00:18.079 UTC
Application Stats: iperf-client-app
   CPU Percentage: 0.11%
   Memory Usage: 720KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 1
Router#
```

Verify Connection between the iPerf Server and iPerf Client Applications

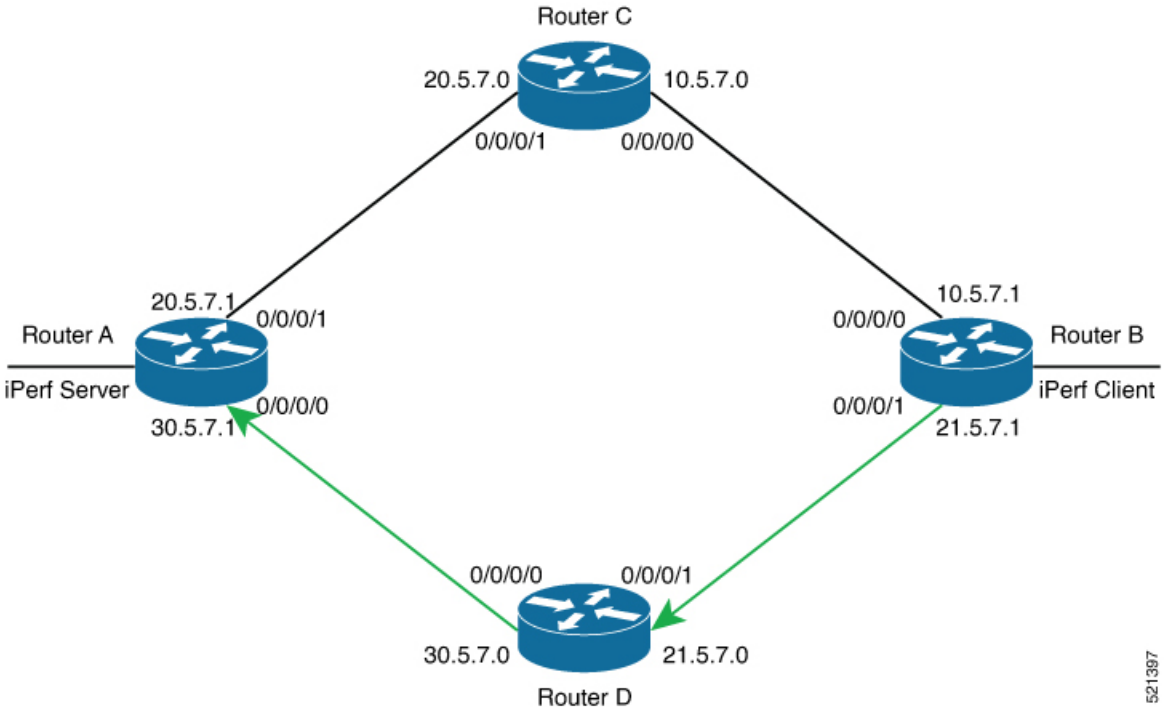
Verify whether the connection is established between iPerf server and iPerf clients by executing the **bash netstat -anput** command on Router A. When the iPerf client is up and running, the entry in the **State** field displays "ESTABLISHED".

```
Router#bash netstat -anput
Thu Dec  3 10:00:33.535 UTC
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:646             0.0.0.0:*               LISTEN      8585/mpls_ldp
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      8567/ssh_server
tcp        0      0 0.0.0.0:830             0.0.0.0:*               LISTEN      8567/ssh_server
tcp6       0      0 :::5201                 :::*                    LISTEN      20829/iperf3
tcp6       0      0 :::22                   :::*                    LISTEN      8567/ssh_server
tcp6       0      0 :::830                  :::*                    LISTEN      8567/ssh_server
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65322         ESTABLISHED 20829/iperf3
```

```
tcp6       0      0 30.5.7.1:5201           100.0.0.9:65302         ESTABLISHED 20829/iperf3
udp        0      0 0.0.0.0:646             0.0.0.0:*                           8585/mpls_ldp
udp        0      0 0.0.0.0:3232            0.0.0.0:*                           6833/pim
udp        0      0 0.0.0.0:3503            0.0.0.0:*                           10762/lspv_server
udp        0      0 0.0.0.0:68              0.0.0.0:*                           10704/xr_dhcpcd
udp        0      0 0.0.0.0:496             0.0.0.0:*                           6833/pim
udp6       0      0 :::3503                 :::*                                10762/lspv_server
```

Measure Network Performance

Step 1 Verify the traffic route from Router B to Router A using the **show ip route** command, on Router B.




```
Router#show ip route 30.5.7.1
Thu Dec  3 10:08:01.859 UTC

Routing entry for 30.5.7.0/31
  Known via "ospf 10", distance 110, metric 2, type intra area
  Installed Dec  3 04:49:22.281 for 05:18:39
  Routing Descriptor Blocks
    21.5.7.0, from 100.0.0.7, via FourHundredGigE0/0/0/1
      Route metric is 2
  No advertising protos.
Router#
```

Step 2 Check the network performance between iPerf client and iPerf server (on Router B and Router A).

You can view the network monitoring parameters by executing the **show appmgr application name iperf-client-app logs** command, on Router B that hosts the iPerf client.

```
Router#show appmgr application name iperf-client-app logs
Tue Dec 1 12:50:27.862 UTC
Connecting to host 30.5.7.1, port 5201
[ 4] local 100.0.0.9 port 61384 connected to 30.5.7.1 port 5201
[ ID] Interval          Transfer     Bandwidth Retr     Cwnd
[ 4] 0.00-1.00 sec 1.05 MBytes     8.82 Mbits/sec 0     80.6 KBytes
[ 4] 1.00-2.00 sec 1.26 MBytes     10.6 Mbits/sec 0     136 KBytes
[ 4] 2.00-3.00 sec 1.18 MBytes     9.90 Mbits/sec 0     191 KBytes
[ 4] 3.00-4.00 sec 1.24 MBytes     10.4 Mbits/sec 0     246 KBytes
[ 4] 4.00-5.00 sec 1.18 MBytes     9.90 Mbits/sec 0     301 KBytes
[ 4] 5.00-6.00 sec 1.37 MBytes     11.5 Mbits/sec 0     362 KBytes
[ 4] 6.00-7.00 sec 1.37 MBytes     11.5 Mbits/sec 0     423 KBytes
[ 4] 7.00-8.00 sec 1.43 MBytes     12.0 Mbits/sec 0     486 KBytes
[ 4] 8.00-9.00 sec 1.30 MBytes     11.0 Mbits/sec 0     547 KBytes
[ 4] 9.00-10.00 sec 1.43 MBytes     12.0 Mbits/sec 0    611 KBytes
[ 4] 10.00-11.00 sec 1.62 MBytes     13.6 Mbits/sec 0   707 KBytes
[ 4] 11.00-12.00 sec 1.62 MBytes     13.6 Mbits/sec 0    875 KBytes
[ 4] 12.00-13.00 sec 1.93 MBytes     16.2 Mbits/sec 0    1.07 MBytes
[ 4] 13.00-14.00 sec 1.68 MBytes     14.1 Mbits/sec 0    1.29 MBytes
[ 4] 14.00-15.00 sec 1.06 MBytes     8.86 Mbits/sec 0    1.56 MBytes
[ 4] 15.00-16.00 sec 891 KBytes     7.30 Mbits/sec 0     1.83 MBytes
[ 4] 16.00-17.00 sec 970 KBytes     7.95 Mbits/sec 0     2.12 MBytes
[ 4] 17.00-18.00 sec 1.24 MBytes     10.4 Mbits/sec 0    2.58 MBytes
[ 4] 18.00-19.00 sec 885 KBytes     7.24 Mbits/sec 0     2.65 MBytes
[ 4] 19.00-20.00 sec 1.55 MBytes     13.0 Mbits/sec 0    3.10 MBytes
[ 4] 20.00-21.00 sec 820 KBytes     6.71 Mbits/sec 0     3.10 MBytes
[ 4] 21.00-22.00 sec 1.72 MBytes     14.4 Mbits/sec 6     2.42 MBytes
[ 4] 22.00-23.00 sec 0.00 Bytes     0.00 bits/sec 5      2.30 MBytes
[ 4] 23.00-24.00 sec 256 KBytes     2.10 Mbits/sec 0     1.35 MBytes
[ 4] 24.00-25.00 sec 1.56 MBytes     13.1 Mbits/sec 237   1.83 MBytes
[ 4] 25.00-26.00 sec 1.90 MBytes     15.9 Mbits/sec 0     2.17 MBytes
[ 4] 26.00-27.00 sec 382 KBytes     3.12 Mbits/sec 61     1.95 MBytes
[ 4] 27.00-28.00 sec 0.00 Bytes     0.00 bits/sec 0     1.39 MBytes
[ 4] 28.00-29.00 sec 3.35 MBytes     28.1 Mbits/sec 0     1.52 MBytes
[ 4] 29.00-30.00 sec 954 KBytes     7.82 Mbits/sec 0     1.58 MBytes
[ 4] 30.00-31.00 sec 1018 KBytes     8.34 Mbits/sec 0     1.64 MBytes
[ 4] 31.00-32.00 sec 1.24 MBytes     10.4 Mbits/sec 0     1.71 MBytes
[ 4] 32.00-33.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.76 MBytes
[ 4] 33.00-34.00 sec 1.61 MBytes     13.5 Mbits/sec 0     1.80 MBytes
[ 4] 34.00-35.00 sec 1.46 MBytes     12.2 Mbits/sec 0     1.82 MBytes
[ 4] 35.00-36.00 sec 1.18 MBytes     9.89 Mbits/sec 0     1.83 MBytes
[ 4] 36.00-37.00 sec 1.36 MBytes     11.4 Mbits/sec 0     1.84 MBytes
[ 4] 37.00-38.00 sec 1.36 MBytes     11.4 Mbits/sec 0     1.84 MBytes
[ 4] 38.00-39.00 sec 1.24 MBytes     10.4 Mbits/sec 0     1.84 MBytes
[ 4] 39.00-40.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.85 MBytes
[ 4] 40.00-41.00 sec 1.25 MBytes     10.5 Mbits/sec 0     1.86 MBytes
[ 4] 41.00-42.00 sec 1.40 MBytes     11.8 Mbits/sec 0     1.88 MBytes
[ 4] 42.00-43.00 sec 1.12 MBytes     9.37 Mbits/sec 0     1.91 MBytes
[ 4] 43.00-44.00 sec 1.12 MBytes     9.40 Mbits/sec 0     1.96 MBytes
[ 4] 44.00-45.00 sec 1.20 MBytes     10.1 Mbits/sec 0     2.02 MBytes
[ 4] 45.00-46.00 sec 1.27 MBytes     10.7 Mbits/sec 0     2.11 MBytes
[ 4] 46.00-47.00 sec 1.30 MBytes     10.9 Mbits/sec 0     2.22 MBytes
[ 4] 47.00-48.00 sec 1.25 MBytes     10.5 Mbits/sec 0     2.36 MBytes
[ 4] 48.00-49.00 sec 1.43 MBytes     12.0 Mbits/sec 0     2.53 MBytes
```

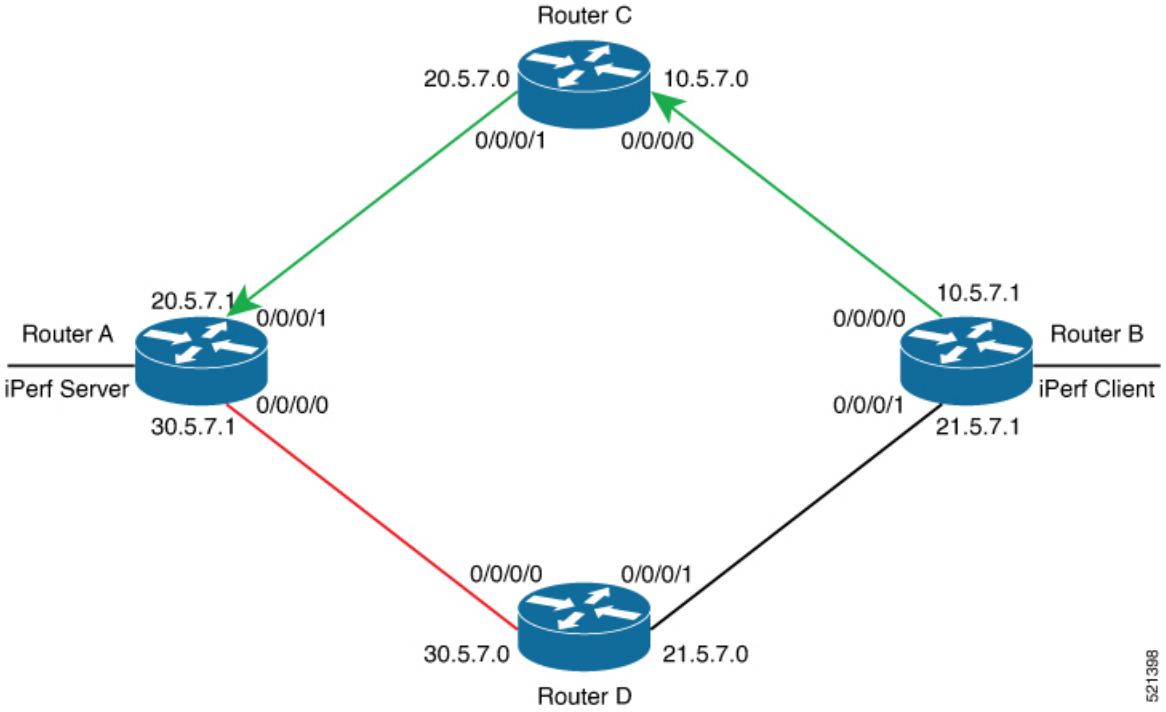
Step 3 Bring down the interface on Router D using the **shut** command to trigger a route update.

```
Router(config)#interface FourhundredGig0/0/0/0
Router(config-if)#shut
Router(config-if)#commit
```

Note Because of the interface shutdown, the route to 30.5.7.1 needs to be updated and hence momentarily there will be no route to this address.

Step 4 During the route update, check the network performance by executing the **show appmgr application name** *app_name* **logs** command.

You will notice that the entries in the **Bandwidth** field is Zero for a short duration, when the new route is installed.




```
Router#show appmgr application name iperf-client-app logs
Tue Dec 1 12:59:40.349 UTC
Connecting to host 30.5.7.1, port 5201
[ 4] local 100.0.0.9 port 61384 connected to 30.5.7.1 port 5201
15
[ ID] Interval          Transfer Bandwidth      Retr     Cwnd
[ 4] 0.00-1.00 sec 1.05 MBytes 8.82 Mbits/sec 0     80.6 KBytes
[ 4] 1.00-2.00 sec 1.26 MBytes 10.6 Mbits/sec 0     136 KBytes
[ 4] 2.00-3.00 sec 1.18 MBytes 9.90 Mbits/sec 0     191 KBytes
[ 4] 3.00-4.00 sec 1.24 MBytes 10.4 Mbits/sec 0     246 KBytes
[ 4] 4.00-5.00 sec 1.18 MBytes 9.90 Mbits/sec 0     301 KBytes
[ 4] 5.00-6.00 sec 1.37 MBytes 11.5 Mbits/sec 0     362 KBytes
[ 4] 6.00-7.00 sec 1.37 MBytes 11.5 Mbits/sec 0     423 KBytes
[ 4] 7.00-8.00 sec 1.43 MBytes 12.0 Mbits/sec 0     486 KBytes
[ 4] 8.00-9.00 sec 1.30 MBytes 11.0 Mbits/sec 0     547 KBytes
[ 4] 9.00-10.00 sec 1.43 MBytes 12.0 Mbits/sec 0     611 KBytes
[ 4] 10.00-11.00 sec 1.62 MBytes 13.6 Mbits/sec 0     707 KBytes
[ 4] 11.00-12.00 sec 1.62 MBytes 13.6 Mbits/sec 0     875 KBytes
[ 4] 12.00-13.00 sec 1.93 MBytes 16.2 Mbits/sec 0     1.07 MBytes
[ 4] 13.00-14.00 sec 1.68 MBytes 14.1 Mbits/sec 0     1.29 MBytes
[ 4] 14.00-15.00 sec 1.06 MBytes 8.86 Mbits/sec 0     1.56 MBytes
[ 4] 15.00-16.00 sec 891 KBytes 7.30 Mbits/sec 0     1.83 MBytes
[ 4] 16.00-17.00 sec 970 KBytes 7.95 Mbits/sec 0     2.12 MBytes
[ 4] 17.00-18.00 sec 1.24 MBytes 10.4 Mbits/sec 0     2.58 MBytes
[ 4] 18.00-19.00 sec 885 KBytes 7.24 Mbits/sec 0     2.65 MBytes
[ 4] 19.00-20.00 sec 1.55 MBytes 13.0 Mbits/sec 0     3.10 MBytes
[ 4] 20.00-21.00 sec 820 KBytes 6.71 Mbits/sec 0     3.10 MBytes
[ 4] 21.00-22.00 sec 1.72 MBytes 14.4 Mbits/sec 6     2.42 MBytes
[ 4] 22.00-23.00 sec 0.00 Bytes 0.00 bits/sec 5     2.30 MBytes
[ 4] 23.00-24.00 sec 256 KBytes 2.10 Mbits/sec 0     1.35 MBytes
```

```
[ 4] 24.00-25.00 sec 1.56 MBytes 13.1 Mbits/sec 237     1.83 MBytes
[ 4] 25.00-26.00 sec 1.90 MBytes 15.9 Mbits/sec 0     2.17 MBytes
[ 4] 26.00-27.00 sec 382 KBytes 3.12 Mbits/sec 61     1.95 MBytes
[ 4] 27.00-28.00 sec 0.00 Bytes 0.00 bits/sec 0     1.39 MBytes
[ 4] 28.00-29.00 sec 3.35 MBytes 28.1 Mbits/sec 0     1.52 MBytes
[ 4] 29.00-30.00 sec 954 KBytes 7.82 Mbits/sec 0     1.58 MBytes
[ 4] 30.00-31.00 sec 1018 KBytes 8.34 Mbits/sec 0     1.64 MBytes
[ 4] 31.00-32.00 sec 1.24 MBytes 10.4 Mbits/sec 0     1.71 MBytes
[ 4] 32.00-33.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.76 MBytes
[ 4] 33.00-34.00 sec 1.61 MBytes 13.5 Mbits/sec 0     1.80 MBytes
[ 4] 34.00-35.00 sec 1.46 MBytes 12.2 Mbits/sec 0     1.82 MBytes
[ 4] 35.00-36.00 sec 1.18 MBytes 9.89 Mbits/sec 0     1.83 MBytes
[ 4] 36.00-37.00 sec 1.36 MBytes 11.4 Mbits/sec 0     1.84 MBytes
[ 4] 37.00-38.00 sec 1.36 MBytes 11.4 Mbits/sec 0     1.84 MBytes
[ 4] 38.00-39.00 sec 1.24 MBytes 10.4 Mbits/sec 0     1.84 MBytes
[ 4] 39.00-40.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.85 MBytes
[ 4] 40.00-41.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.86 MBytes
[ 4] 41.00-42.00 sec 1.40 MBytes 11.8 Mbits/sec 0     1.88 MBytes
[ 4] 42.00-43.00 sec 1.12 MBytes 9.37 Mbits/sec 0     1.91 MBytes
[ 4] 43.00-44.00 sec 1.12 MBytes 9.40 Mbits/sec 0     1.96 MBytes
[ 4] 44.00-45.00 sec 1.20 MBytes 10.1 Mbits/sec 0     2.02 MBytes
[ 4] 45.00-46.00 sec 1.27 MBytes 10.7 Mbits/sec 0     2.11 MBytes
[ 4] 46.00-47.00 sec 1.30 MBytes 10.9 Mbits/sec 0     2.22 MBytes
[ 4] 95.00-96.00 sec 1.48 MBytes 12.4 Mbits/sec 0     1.82 MBytes
[ 4] 96.00-97.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.83 MBytes
[ 4] 97.00-98.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.83 MBytes
[ 4] 98.00-99.00 sec 1.49 MBytes 12.5 Mbits/sec 0     1.84 MBytes
[ 4] 99.00-100.00 sec 1.25 MBytes 10.5 Mbits/sec 0     1.86 MBytes
[ 4] 100.00-101.00 sec 1.21 MBytes 10.2 Mbits/sec 0     1.89 MBytes
[ 4] 101.00-102.00 sec 1.34 MBytes 11.2 Mbits/sec 0     1.94 MBytes
[ 4] 102.00-103.00 sec 1.25 MBytes 10.5 Mbits/sec 0     2.01 MBytes
[ 4] 103.00-104.00 sec 1.30 MBytes 10.9 Mbits/sec 0     2.09 MBytes
[ 4] 104.00-105.00 sec 1.25 MBytes 10.5 Mbits/sec 0     2.17 MBytes
[ 4] 105.00-106.00 sec 1.39 MBytes 11.6 Mbits/sec 0     2.33 MBytes
[ 4] 106.00-107.00 sec 1.01 MBytes 8.47 Mbits/sec 0     2.46 MBytes
[ 4] 107.00-108.00 sec 526 KBytes 4.31 Mbits/sec 0     2.54 MBytes
[ 4] 108.00-109.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 109.00-110.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 110.00-111.00 sec 0.00 Bytes 0.00 bits/sec 0     2.54 MBytes
[ 4] 111.00-112.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 112.00-113.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 113.00-114.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 114.00-115.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 115.00-116.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 116.00-117.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 117.00-118.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 118.00-119.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 119.00-120.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 120.00-121.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 121.00-122.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 122.00-123.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 123.00-124.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 124.00-125.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 125.00-126.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 126.00-127.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 127.00-128.00 sec 0.00 Bytes 0.00 bits/sec 1     1.41 KBytes
[ 4] 128.00-129.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 129.00-130.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 130.00-131.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 131.00-132.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 132.00-133.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 133.00-134.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 134.00-135.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
```

```
[ 4] 135.00-136.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 136.00-137.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 137.00-138.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 138.00-139.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 139.00-140.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 140.00-141.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 141.00-142.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 142.00-143.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 143.00-144.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 144.00-145.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 145.00-146.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 146.00-147.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 147.00-148.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 148.00-149.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 149.00-150.00 sec 0.00 Bytes 0.00 bits/sec 0     1.41 KBytes
[ 4] 150.00-151.00 sec 700 KBytes 5.73 Mbits/sec 847   600 KBytes
[ 4] 151.00-152.00 sec 954 KBytes 7.82 Mbits/sec 993   1.32 MBytes
[ 4] 152.00-153.00 sec 509 KBytes 4.17 Mbits/sec 0     1.79 MBytes
[ 4] 153.00-154.00 sec 1.08 MBytes 9.07 Mbits/sec 0    1.85 MBytes
[ 4] 154.00-155.00 sec 1.38 MBytes 11.6 Mbits/sec 0    1.90 MBytes
[ 4] 155.00-156.00 sec 1.55 MBytes 13.0 Mbits/sec 0    1.98 MBytes
[ 4] 156.00-157.00 sec 1.16 MBytes 9.71 Mbits/sec 0    2.04 MBytes
[ 4] 157.00-158.00 sec 1.21 MBytes 10.2 Mbits/sec 0    2.10 MBytes
[ 4] 158.00-159.00 sec 1.26 MBytes 10.6 Mbits/sec 0    2.17 MBytes
[ 4] 159.00-160.00 sec 1.14 MBytes 9.56 Mbits/sec 0    2.23 MBytes
[ 4] 160.00-161.00 sec 1.29 MBytes 10.8 Mbits/sec 0    2.27 MBytes
[ 4] 161.00-162.00 sec 1.24 MBytes 10.4 Mbits/sec 0    2.34 MBytes
[ 4] 162.00-163.00 sec 1.42 MBytes 11.9 Mbits/sec 0    2.41 MBytes
[ 4] 163.00-164.00 sec 1.11 MBytes 9.34 Mbits/sec 0    2.46 MBytes
[ 4] 164.00-165.00 sec 1.39 MBytes 11.7 Mbits/sec 0    2.56 MBytes
[ 4] 165.00-166.00 sec 995 KBytes 8.16 Mbits/sec 0     2.69 MBytes
[ 4] 166.00-167.00 sec 1.88 MBytes 15.7 Mbits/sec 0    2.94 MBytes
[ 4] 167.00-168.02 sec 950 KBytes 7.69 Mbits/sec 0     3.12 MBytes
[ 4] 168.02-169.00 sec 1.79 MBytes 15.2 Mbits/sec 0    3.12 MBytes
[ 4] 169.00-170.01 sec 1.27 MBytes 10.6 Mbits/sec 0    3.12 MBytes
[ 4] 170.01-171.00 sec 1.25 MBytes 10.5 Mbits/sec 23    1.60 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-600.00 sec   704 MBytes  9.84 Mbits/sec  12069   sender
[  4]   0.00-600.00 sec   702 MBytes  9.82 Mbits/sec          receiver

iperf Done.

<!-On Router A!>
Router#show appmgr application name iperf-server-app stats
Thu Dec  3 11:45:47.790 UTC
Application Stats: iperf-server-app
   CPU Percentage: 0.00%
   Memory Usage: 816KiB / 31.23GiB
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
   Block IO: 0B / 0B
   PIDs: 1
<!-On Router B!>
Router#show appmgr application name iperf-client-app stats
Thu Dec  3 11:45:59.418 UTC
Application Stats: iperf-client-app
   CPU Percentage: 0.00%
   Memory Usage: 0B / 0B
   Memory Percentage: 0.00%
   Network IO: 0B / 0B
```

```
Block IO: 0B / 0B
PIDs: 0
```

Stop iPerf Applications

Stop the iPerf applications on Router A and Router B using the **appmgr application stop name** *app_name* command. The **application stop** command can only be used for applications that are registered, activated, and are currently running. The **application stop** command stops only the application and does not clean up the resources used by the application.

You can verify the status of the application using the **show appmgr application-table** command. The **Status** is displayed as **Exited** if the application has been stopped successfully.

```
Router#appmgr application stop name iperf-server-app
Mon Nov 30 13:38:36.202 UTC
Router#show appmgr application-table
Mon Nov 30 13:38:36.999 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   Exited (1) Less than a se
Router#
```

Start iPerf Applications

Start or restart an application that has been stopped (and not deactivated) using the **appmgr application start name** *app_name* command.

```
Router#appmgr application start name iperf-server-app
Tue Dec 1 13:06:21.996 UTC
Router#show appmgr application-table
Mon Nov 30 13:38:36.999 UTC
Name                 Type    Config State  Status
--------             ------- ------------- -------------------------
iperf-server-app     Docker    Activated   UP(1) Less than a second
Router#
```

Deactivate iPerf Applications

Step 1 Deactivate the iPerf applications using the **no appmgr application** *app_name* command. You deactivate the installed application when you want to release all resources used by the application.

```
Router#config
Router(config)#no appmgr application iperf-server-app
Router(config)#commit
```

Step 2 Verify the status of the application by using the **show appmgr application-table** *app_name* **stats** command.

```
Router#show appmgr application-table
Mon Nov 30 13:39:51.197 UTC
Router#
```

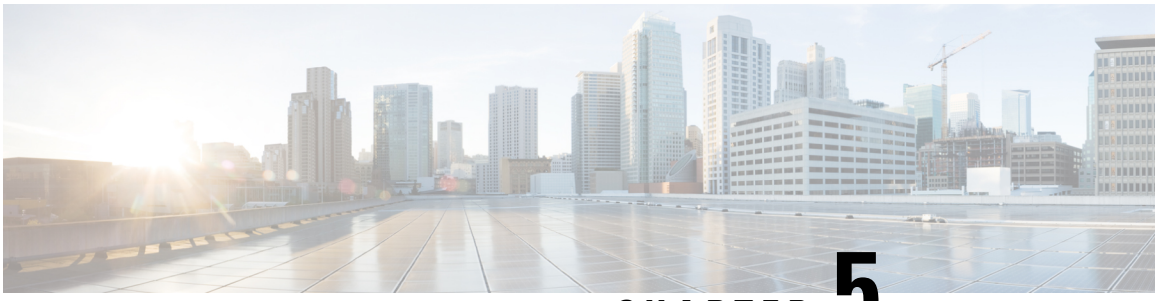
Note You can activate a deactivated application using the **appmgr application** *app_name* **activate type docker source** *source_name* command.

Uninstall iPerf Applications

Uninstall the applications using the **appmgr package uninstall package** *package_name* command.

After the application is successfully uninstalled, executing the **show appmgr source-table** command displays no result.

```
Router#appmgr package uninstall package iperf
table
Mon Nov 30 13:41:05.155 UTC
Router#show appmgr source-table
Mon Nov 30 13:41:05.936 UTC
Router#
```



CHAPTER 5

Cisco Secure DDoS Edge Protection

Table 2: Feature History Table

Feature Name	Release Information	Description
Cisco Secure DDoS Edge Protection	Release 7.10.1	Cisco Secure DDoS Edge Protection solution provides protection from Denial-of-Service (DDoS) attacks and helps to mitigate them. DDoS Edge Protection solution helps you to: • Reduce the total cost of ownership for the DDoS solution by reducing the overall scrubbing capacity requirement which in turn reduces the overall power consumption, cooling, and other maintenance costs. • Improve customer satisfaction and helps in achieving SLAs due to the reduced attack detection and response time.

Cisco Secure DDoS Edge Protection is a software solution that stops DDoS attacks at the ingress side of the service provider network edge on the NCS 540 platform.

With the help of the DDoS Edge Protection solution, you can detect the DDoS attacks and perform mitigation actions on the router. For the detection services at the edge network, you must configure the following entities:

- DDoS Edge Protection Controller—Manages and monitors the Detector docker application and mitigates the attacks. The controller manages a distributed network of edge detectors, analyzes the detection trends across the network, and orchestrates the cross-network visibility and mitigation. The controller also delivers a complete system management lifecycle for the entire service.
- DDoS Edge Protection Detector—A real-time DDoS detection microservice container application that runs as a docker-application on a router with the DDoS controller. The DDOS controller can run on a cloud, server, or customer premises, and is connected to this application.

The NCS 540 platform supports traffic detection on the GPRS Tunneling Protocol User Plane (GTP-U). You can select on which interface the traffic must be monitored. When the protection software solution is implemented, the GTP traffic flow is filtered, and a DDoS attack is detected.

When the DDoS attack is detected, the DDoS Edge Protection Controller pushes the mitigation action to the NCS 540 router through the detector.

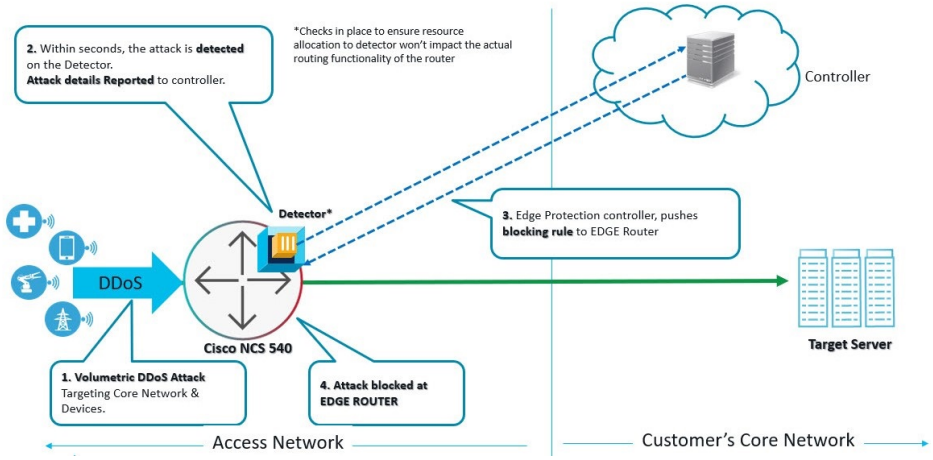
This whole process is performed through an Access Control List (ACL) configuration. A unique GTP flow is identified with a unique Tunnel Endpoint Identifier (TEID) present in the GTP header. The GTP header follows the UDP header. The identified flow that is associated with a TEID can be mitigated with the help of an ACE using the UDF option (User-defined filter) configuration in the ACL.

Note

- Only GTP-U traffic is monitored. The detection is not supported for the GTP-C traffic.
- If the GTP header is incorrect, then the sampled record is discarded.

For more information on the supported platforms, see Supported NCS 540 Platforms.

The following sample topology diagram shows that the NCS 540 series router runs the DDoS Edge Protection detector. The DDoS Edge Protection controller is running as an external server that manages the DDoS Edge Protection detector.



- Prerequisites for Installing DDoS Edge Protection, on page 38
- Restrictions of DDoS Edge Protection Solution, on page 39
- Supported NCS 540 Platforms, on page 39
- Install and Configure DDoS Edge Protection, on page 40
- Verify DDoS Edge Protection Application Configuration, on page 42

Prerequisites for Installing DDoS Edge Protection

- Configure the management interface to reach the DDoS controller IP Address.
- Manually configure the base ACL, UDF, NetFlow, and SSH configurations.

  For more information, see Install and Configure DDoS Edge Protection, on page 40.

- Use the following configuration to mitigate the DDoS attack as UDF is a qualifier for ACL:

```
hw-module profile tcam format access-list ipv4 src-addr dst-addr
src-port dst-port proto frag-bit enable-capture udf1 udf-gtp location
<location>
```

- Reload the router (as a hw-module profile configuration is performed).

Restrictions of DDoS Edge Protection Solution

- Only the IPv4 GTP tunnel is supported. The inner traffic can be both IPv4 or IPv6.
- The default VRF configuration is only supported on the management port and the non-default VRF is not supported. Ensure that you configure the management port only on a default VRF so that there is communication between the docker and the controller.

Supported NCS 540 Platforms

From Cisco IOS XR 7.7.1, record type GTP is supported on the NCS 540 platform. For example:

```
flow monitor-map DetectPro_Monitor_IPV4
  record ipv4 gtp
```

You can perform DDoS Edge Protection on the following NCS 540 platforms:

Table 3: Supported NCS 540 Platforms

IOS XR Release	Platform
7.7.1	N540-ACC-SYS
	N540X-ACC-SYS
	N540-24Z8Q2C-SYS
7.10.1	N540-24Q8L2DD-SYS
	N540X-16Z4G8Q2C-D/A
	N540X-16Z8Q2C-D
	N540-28Z4C-SYS-D/A
	N540X-12Z16G-SYS-D/A
	N540X-12Z20G-SYS-D/A

Unsupported NCS 540 Platforms

The following NCS platforms are not supported:

- N540X-4Z14G2Q-D/A
- N540X-8Z16G-SYS-D/A

- N540X-6Z18G-SYS-D/A
- N540-6Z18G-SYS-D/A
- N540-6Z14G-SYS-D
- N540H-FH-AGG-SYS
- N540-FH-CSR-SYS

Install and Configure DDoS Edge Protection

You can install the DDoS Edge Protection application through the DDoS edge protection controller. Perform the following:

1. Install and download the DDoS Edge Protection Controller Software package from the Software Download page. You can access the user interface, when the controller installation is complete.

   Log in to the controller services instance to monitor, manage, and control the device.

2. Perform the following base configurations such as ACL, UDF, hw-module, NetFlow configuration, and SSH manually on the NCS 540 router:

   Configure UDF

   ```
   router(config)#udf udf-gtp header outer l4 offset 12 length 4
   ```

   The user-defined field, allows you to define a custom key by specifying the location and size of the field to match.

   For example, this command helps in matching the TEID value in the GTP header which is a 4-byte value at the 12th offset from the outer L4 header (UDP L4 header).

   Configure the hardware module or TCAM

   ```
   router(config)#hw-module profile tcam format access-list ipv4 src-addr dst-addr src-port
    dst-port proto frag-bit enable-capture udf1 udf-gtp location <location>
   ```

   Reload the router (as hw-module profile and UDF configuration is performed).

   Configure Loopback

   ```
   RP/0/RP0/CPU0:ios(config)#interface Loopback100
   RP/0/RP0/CPU0:ios(config-if)# ipv4 address 10.1.1.1 255.255.255.255
   RP/0/RP0/CPU0:ios(config)#interface Loopback101
   RP/0/RP0/CPU0:ios(config-if)# ipv4 address 10.10.10.2 255.255.255.255
   RP/0/RP0/CPU0:ios(config-if)#
   ```

   Configure ACL

   ```
   RP/0/RP0/CPU0:ios(config)#ipv4 access-list gtp
   RP/0/RP0/CPU0:ios(config-ipv4-acl)# 2000 permit udp any any eq 2152 capture
   RP/0/RP0/CPU0:ios(config-ipv4-acl)# 2010 permit ipv4 any any
   ```

Note Ensure that you configure the ACL name as *gtp*. This option is applicable only for Cisco IOS XR 7.7.1 or later.

For more information on implementing access lists and prefix lists, see Understanding Access Lists.

If there is any DDoS attack, the controller performs the mitigation action using the ACL rule automatically. The controller pushes the ACL deny rules to block the traffic coming with the DDoS attacker TEID values.

Note The GTP TEID value of 0x1 varies based on the GTP Tunnel ID.

The following is a sample configuration to deny the ACE rule that denies DDoS attacker traffic with TEID value of 0x1:

```
10 deny ipv4 any any udf udf-gtp 0x1 0xffffffff
```

The controller pushes the configuration to the router.

Configure SSH

```
router(config)#ssh server v2
router(config)#ssh server netconf
router(config)#netconf agent tty
router(config)#netconf-yang agent ssh
router(config)#netconf agent tty
!
router(config)#ssh timeout 120
router(config)#ssh server rate-limit 600
router(config)#ssh server session-limit 110
router(config)#ssh server v2
router(config)#ssh server vrf default
router(config)#ssh server netconf vrf default
```

To configure TPA (until 7.8.1), perform the following steps:

```
RP/0/RP0/CPU0:ios(config)#tpa
RP/0/RP0/CPU0:ios(config-tpa)#vrf default
RP/0/RP0/CPU0:ios(config-tpa-vrf)#east-west Loopback101
RP/0/RP0/CPU0:ios(config-tpa-vrf)#address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#default-route mgmt
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#update-source dataports Loopback100
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)#
```

To configure TPA from 7.9.1 or later on NCS 540, perform the following steps:

```
RP/0/RP0/CPU0:ios(config)#linux networking
RP/0/RP0/CPU0:ios(config-lnx-net)#vrf default
RP/0/RP0/CPU0:ios(config-lnx-vrf)#east-west Loopback101
RP/0/RP0/CPU0:ios(config-lnx-vrf)#address-family ipv4
RP/0/RP0/CPU0:ios(config-lnx-af)#default-route software-forwarding
RP/0/RP0/CPU0:ios(config-lnx-af)#source-hint default-route interface MgmtEth0/RP0/CPU0/0
RP/0/RP0/CPU0:ios(config-lnx-af)#
```

3. Reload the router (as the hw-module profile configuration is performed).
4. On the NCS 540 router, check the device connection to the DDoS controller using the **ping** command.

```
RP/0/RP0/CPU0:Router#ping 10.105.237.54
Thu Jun  1 07:16:43.654 UTC
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.105.237.54 timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 2/2/4 ms
RP/0/RP0/CPU0:Router#bash
```

```
Thu Jun  1 07:16:53.024 UTC
[Router:~]$ping 10.105.237.54
PING 10.105.237.54 (10.105.237.54) 56(84) bytes of data.
64 bytes from 10.105.237.54: icmp_seq=1 ttl=63 time=1.73 ms
64 bytes from 10.105.237.54: icmp_seq=2 ttl=63 time=1.29 ms
64 bytes from 10.105.237.54: icmp_seq=3 ttl=63 time=1.27 ms
64 bytes from 10.105.237.54: icmp_seq=4 ttl=63 time=1.75 ms
^C
--- 10.105.237.54 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.270/1.510/1.751/0.230 ms
[Router:~]$
```

5. Add device details on the controller panel and ensure that all the three indicators (Deployment, Container, and Configuration) are green.

   For more information on installing the DDoS controller, see the DDoS Edge Protection Installation guide.

For more information on the DDoS Edge Protection, see Cisco Secure DDoS Edge Protection Data Sheet.

Verify DDoS Edge Protection Application Configuration

You can also verify if the DDoS controller pushes the CLI to the device using the following **show running-config** commands on the device:

```
RP/0/RP0/CPU0:Router#show running-config appmgr
Thu Jun  1 07:33:36.741 UTC
appmgr
 application esentryd
  activate type docker source esentryd-cisco-20230431633 docker-run-opts "-p 10000:10000/tcp
 -p 5005:5005/udp --env-file /harddisk:/ENV_6478443711ac6830700d1aeb --net=host"
 !
!
```

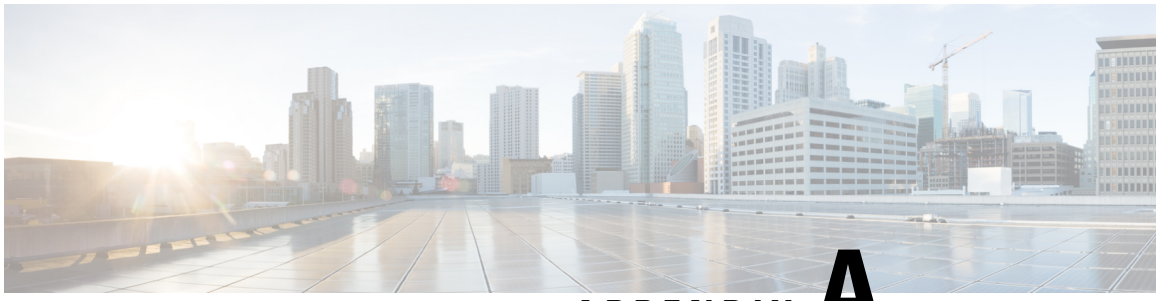
```
RP/0/RP0/CPU0:Router#show running-config flow monitor-map DetectPro_Monitor_IPV4
Thu Jun  1 07:34:34.236 UTC
flow monitor-map DetectPro_Monitor_IPV4
 record ipv4 gtp
 option filtered
 exporter DetectPro_GPB
 cache entries 1000000
 cache timeout active 1
 cache timeout inactive 1
 cache timeout rate-limit 100000
!
```

```
RP/0/RP0/CPU0:Router#show running-config flow exporter-map DetectPro_GPB
Thu Jun  1 07:35:10.417 UTC
flow exporter-map DetectPro_GPB
 version protobuf
 !
 transport udp 5005
 source TenGigE0/0/0/16
 destination 10.1.1.2
!
```

```
RP/0/RP0/CPU0:Router#show running-config interface tenGigE 0/0/0/16
Thu Jun  1 07:35:25.778 UTC
interface TenGigE0/0/0/16
 shutdown
 flow ipv4 monitor DetectPro_Monitor_IPV4 sampler DetectPro_SMPL ingress
```

```
 ipv4 access-group gtp ingress
!

RP/0/RP0/CPU0:Router#show appmgr application-table
Thu Jun  1 07:36:21.692 UTC
Name     Type   Config State Status
-------- ------ ------------ ------------------------------------------------
esentryd Docker  Activated   Up 8 minutes
RP/0/RP0/CPU0:Router#
```



APPENDIX A

Accessing the Networking Stack

This section is applicable for Cisco IOS XR Release 7.9.1 and earlier for the following variants:

- N540-ACC-SYS
- N540X-ACC-SYS
- N540-24Z8Q2C-SYS

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.

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.

To have an application 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.

Configure the Source 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. 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.

External communication is successfully enabled on IOS XR.

East-West Communication for Third-Party Applications

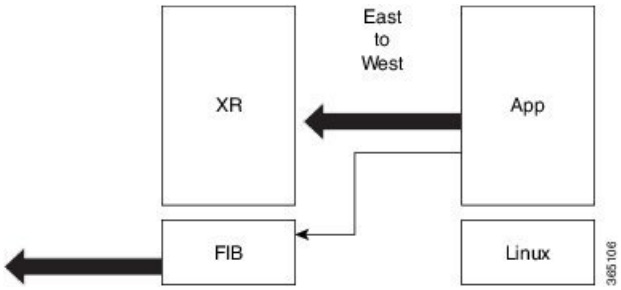
East-West communication on IOS XR is a mechanism by which applications hosted in containers interact with native XR applications (hosted in the XR control plane).

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 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.

Figure 5: 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
```

2. Configure another Loopback interface that will be the East interface. You must configure this loopback interface to act as the TPA-facing interface, and Cisco IOS XR interacts with the TPA using this interface.

```
RP/0/RP0/CPU0:ios(config)# interface Loopback100
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 15.1.1.1/32
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
```

3. Verify the creation of the Loopback1 (West) and Loopback100 (East) interfaces.

```
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
Loopback100                    15.1.1.1        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
```

4. Configure the TPAs.

```
RP/0/RP0/CPU0:ios(config)#tpa vrf default east-west loopback 1
RP/0/RP0/CPU0:ios(config)#tpa vrf default address-family ipv4 default-route mgmt
RP/0/RP0/CPU0:ios(config)#tpa vrf default address-family ipv4 update-source dataports
loopback 100
RP/0/RP0/CPU0:ios(config)#commit
```

5. Verify that a TPA interface that sets up Loopback100 as the East interface is configured.

```
RP/0/RP0/CPU0:ios#sh run tpa
Mon Jun 7 07:22:08.324 UTC
tpa
vrf default
east-west Loopback1
address-family ipv4
default-route mgmt
```

```
update-source dataports Loopback100
!
!
!
```

6. Verify the E-W communication configuration by logging into the container and checking the routes. You can also ping the router-side East interface.

Note You can use the bash command to connect to the router and execute commands only in the testing environment.

```
RP/0/RP0/CPU0:ios#bash
Mon Jun 7 07:22:57.650 UTC
[ios:~]$ docker exec -it a0 bash
root@host:0_RP0:/# ip route
default dev fwd_ew scope link src 15.1.1.1
8.8.8.8 dev fwd_ew scope link src 15.1.1.1
192.168.104.0/24 dev Mg0_RP0_CPU0_0 scope link src 192.168.104.10
root@host:0_RP0:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=255 time=0.397 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=0.535 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 0.397/0.466/0.535/0.069 ms
root@host:0_RP0:/#
```

For more information on how to launch your own containers, see Using Docker for Hosting Applications on Cisco IOS XR topic. .

Configuring Multiple VRFs for Application Hosting

Cisco NCS 540 routers support the configuration of multiple VRFs. The applications hosted in third-party containers can communicate with VRFs configured on XR, after east-west communication has been enabled on the VRFs.

This section describes the configuration for creating mulitple VRFs, and enabling east-west communication between the applications and the VRFs.

Configuration Procedure

Use the following steps to configure multiple VRFs for use on Cisco IOS XR.

1. Configure VRFs on XR.

```
RP/0/RP0/CPU0:ios(config)# vrf purple
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-vrf)# exit

RP/0/RP0/CPU0:ios(config)# vrf green
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-vrf)# exit
```

```
RP/0/RP0/CPU0:ios(config)# telnet vrf purple ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf purple ipv6 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf green ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet vrf green ipv6 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet ipv4 server max-servers 2
RP/0/RP0/CPU0:ios(config)# telnet ipv6 server max-servers 2
```

2. Configure the interfaces to be used with the VRFs.

```
RP/0/RP0/CPU0:ios(config)# interface loopback1
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 1.1.1.1 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 10::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface loopback2
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 2.2.2.2 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 20::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface loopback3
RP/0/RP0/CPU0:ios(config-if)#vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 3.3.3.3 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 30::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface loopback4
RP/0/RP0/CPU0:ios(config-if)# vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 4.4.4.4 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 40::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface mgmtEth 0/RP0/CPU0/0
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address dhcp
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-if)# vrf purple
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 10.20.30.40 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 24::1/64
RP/0/RP0/CPU0:ios(config-if)# exit

RP/0/RP0/CPU0:ios(config)# interface gigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-if)# vrf green
RP/0/RP0/CPU0:ios(config-if)# ipv4 address 40.30.20.10 255.255.255.0
RP/0/RP0/CPU0:ios(config-if)# ipv6 address 22::1/64
RP/0/RP0/CPU0:ios(config-if)# exit
RP/0/RP0/CPU0:ios(config)# commit
Fri Sep  1 12:04:37.796 UTC
```

3. Configure TPA VRFs.

```
RP/0/RP0/CPU0:ios(config)# tpa
RP/0/RP0/CPU0:ios(config-tpa)# vrf purple
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback1
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback2
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source GigabitEthernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
```

```
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source Gigabitethernet 0/0/0/0
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# exit


RP/0/RP0/CPU0:ios(config-tpa)# vrf green
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback3
RP/0/RP0/CPU0:ios(config-tpa-vrf)# east-west loopback4
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv4
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# address-family ipv6
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# update-source Gigabitethernet 0/0/0/1
RP/0/RP0/CPU0:ios(config-tpa-vrf-afi)# exit
RP/0/RP0/CPU0:ios(config-tpa-vrf)# exit
RP/0/RP0/CPU0:ios(config-tpa)# exit
```

4. Validate the configuration.

```
RP/0/RP0/CPU0:ios(config)# show run
Fri Sep  1 12:06:35.596 UTC
...
vrf purple
address-family ipv4
address-family ipv6
vrf green
address-family ipv4
address-family ipv6

telnet vrf green ipv4 server max-servers 2
telnet vrf green ipv6 server max-servers 2
telnet vrf purple ipv4 server max-servers 2
telnet vrf purple ipv6 server max-servers 2
telnet vrf default ipv4 server max-servers 2
telnet vrf default ipv6 server max-servers 2
...
!
tpa
 vrf purple
  east-west loopback1
  east-west loopback2
  address-family ipv4
   update-source GigabitEthernet0/0/0/0
 !
  address-family ipv6
   update-source GigabitEthernet0/0/0/0
 !

 vrf green
  east-west loopback3
  east-west loopback4
  address-family ipv4
   update-source GigabitEthernet0/0/0/1
 !
  address-family ipv6
   update-source GigabitEthernet0/0/0/1
 !
!
interface loopback1
 vrf purple
 ipv4 address 1.1.1.1 255.255.255.0
 ipv6 address 10::1/64
!
```

```
interface loopback2
 vrf purple
 ipv4 address 2.2.2.2 255.255.255.0
 ipv6 address 20::1/64
!
interface loopback3
 vrf green
 ipv4 address 3.3.3.3 255.255.255.0
 ipv6 address 30::1/64
!
interface loopback4
 vrf green
 ipv4 address 4.4.4.4 255.255.255.0
 ipv6 address 40::1/64
!
interface MgmtEth0/RP0/CPU0/0
 vrf purple
 ipv4 address dhcp
!
router static
 address-family ipv4 unicast
  0.0.0.0/0 MgmtEth0/RP0/CPU0/0 10.0.2.2
 !
!
```

You have successfully configured multiple VRFs for use on Cisco IOS XR.

Accessing the Networking Stack