Routing Configuration Guide for Cisco 8000 Series Routers, Cisco IOS XR Release

Cisco 8000 Series Routers
Release: Cumulative | Updated June 30, 2026

Topics included

1 Getting Started

Outlines the release history and update summary for the Routing Configuration Guide. It enables users to access the latest IOS XR features and release information through a centralized, continuously updated resource.

This cumulative guide provides a single, continuously updated version that includes all the latest IOS XR features and release updates. It simplifies your experience by letting you bookmark one link and access the complete guide, instead of navigating through multiple release-specific versions.

Specific changes or updates tied to individual releases are clearly called out within the relevant sections. For a list of features introduced in a specific release, refer to the Release Notes.

The table lists the release numbers for which this document has been updated since its initial publication.

Table 1: Changes to this document

Date	Summary
June 2026	First published for Release 26.2.1

2 Implement and Monitor RIB

Topics:

Explains how RIB stores best routes from multiple protocols, selects the overall preferred routes by administrative distance, and supports forwarding and monitoring across the system.

Routing Information Base (RIB) is a distributed collection of routing connectivity information that

- maintains routing information for each router in the network
- stores the best routes learned from all routing protocols that run on the system, and
- selects the best overall routes and downloads them to the line cards for packet forwarding.

How RIB selects routes

Each routing protocol selects its own best routes and installs those routes, along with their attributes, in RIB.

Within a routing protocol, route selection depends on the metric that the protocol uses. The protocol then downloads its best routes, including equal-cost routes, to RIB.

RIB stores routes from all routing protocols and selects the best overall route by comparing the administrative distance of each protocol.

How RIB supports forwarding

After RIB selects the best overall routes, it downloads those routes to the line cards for packet forwarding.

The term RIB refers both to the RIB processes and to the collection of route data that RIB contains.

Programmatic configuration support

You can programmatically configure RIB for Border Gateway Protocol (BGP) and retrieve operational data by using the `openconfig-rib-bgp.yang` OpenConfig data model.

Prerequisites for implementing RIB

Lists the user access and base software requirements needed before implementing RIB on the router.

Use this reference to confirm that the required user permissions and base software components are available before you implement RIB.

User access requirements

You must be in a user group that is associated with a task group that includes the required task IDs.

The command reference guides list the task IDs that are required for each command.

If you suspect that your user group assignment prevents you from using a command, contact your Authentication, Authorization, and Accounting (AAA) administrator.

Software requirements

RIB is distributed with the base Cisco IOS XR software and does not require any separate installation.

These are the requirements for base software installation:

- Router
- Cisco IOS XR software
- Base package

RIB key concepts

Explains key RIB concepts, including route selection, protocol data structures, administrative distance, IPv4 support, statistics, quarantining, and route consistency checking.

This topic provides the necessary conceptual background to implement the Cisco RIB feature effectively within a network environment.

How RIB stores and selects routes

Describes how routing protocols install their best routes in RIB and how RIB selects the overall best routes for forwarding.

Each routing protocol selects its own best routes and installs those routes, along with their attributes, in RIB. RIB stores routes from all routing protocols and selects the best routes from among them. After RIB selects the best routes, it downloads them to the line cards for packet forwarding. The term RIB refers both to the RIB processes and to the collection of route data that RIB contains.

Within a routing protocol, route selection depends on the metric that the protocol uses. A protocol downloads its best routes, including routes with the lowest metric or tied metric, to RIB.

RIB selects the best overall route by comparing the administrative distance of the associated routing protocol.

RIB data structures in BGP and other protocols

Explains how RIB data structures differ from protocol-specific route stores such as BRIB and how selected routes are passed to FIB processes for forwarding.

This topic explains how RIB manages data structures independently of other routing protocols and how it interacts with the Forwarding Information Base (FIB).

RIB and protocol-specific data structures

RIB uses processes and maintains data structures that are distinct from those used by other routing applications, such as Border Gateway Protocol (BGP) and other unicast routing protocols.

These routing protocols can use internal data structures that are similar to those used by RIB, and they can refer to those internal data structures as a RIB.

For example, BGP routes are stored in the BGP RIB (BRIB). RIB processes are not responsible for the BRIB. BGP manages the BRIB.

RIB and forwarding tables

The table that the line cards and route processor (RP) use to forward packets is called the Forwarding Information Base (FIB).

RIB processes do not build the FIBs. Instead, RIB sends the selected best routes to the FIB processes through the Bulk Content Downloader (BCDL) process on each line card.

After the routes are downloaded, the FIBs are constructed.

RIB administrative distance

Describes how RIB uses administrative distance to choose between routes of equal prefix length learned from different protocols.

This topic explains the role of administrative distance in route selection and provides the default values for common routing protocols.

How RIB uses longest prefix match

Packet forwarding uses the longest prefix match.

For example, if a packet is destined for `10.0.2.1`, the route `10.0.2.0/24` is preferred over `10.0.0.0/16` because the `/24` prefix is longer and more specific than the `/16` prefix.

How RIB uses administrative distance

When routes from different protocols have the same prefix and prefix length, RIB selects the route by using administrative distance.

For example, Open Shortest Path First (OSPF) has a default administrative distance of `110`, and Intermediate System-to-Intermediate System (IS-IS) has a default administrative distance of `115`. If OSPF and IS-IS both download `10.0.1.0/24` to RIB, RIB prefers the OSPF route because OSPF has the lower administrative distance.

Administrative distance is used only to choose between multiple routes that have the same prefix length.

Table 2: Default administrative distances

Protocol	Default administrative distance
Connected or local routes	0
Static routes	1
External BGP routes	20
OSPF routes	110
IS-IS routes	115
Internal BGP routes	200

Changing administrative distance

You can change the administrative distance for some routing protocols, such as IS-IS, OSPF, and BGP. See the protocol-specific documentation for the correct procedure.

> **Note**
>
> Changing the administrative distance of a protocol on some routers but not on all routers can cause routing loops and other undesirable behavior. This practice is not recommended.

RIB support for IPv4

Explains that Cisco IOS XR RIB supports unicast IPv4 routing and identifies the default RIB tables and process behavior.

This topic provides an overview of IPv4 unicast routing support in the RIB and details the process placement functionality for the RIB manager.

IPv4 routing-table support

In Cisco IOS XR software, RIB tables support unicast routing.

The default routing tables for Cisco IOS XR software RIB are the unicast RIB tables for IPv4 routing.

RIB process placement

The `rib_mgr` RIB process runs on the route processor (RP) card.

If process placement functionality is available and the router supports multiple RPs, RIB processes can be placed on any available node.

RIB statistics

Describes how RIB tracks message and request counters exchanged with clients to help monitor route operations, registrations, notifications, and results.

Use this reference to understand the message counters and request results that RIB maintains for client interactions.

RIB message statistics

RIB supports statistics for messages, or requests, that flow between RIB and its clients.

Protocol clients send messages to RIB, such as route add, route delete, and next-hop registration requests. RIB also sends messages such as redistributed routes, advertisements, and next-hop notifications.

These statistics show what messages were sent and how many messages were sent between the RIB server and its clients.

Use the **show rib statistics** command to display these statistics.

Client request counters

RIB maintains counters for all requests that clients send, including these request types:

- Route operations
- Table registrations
- Next-hop registrations
- Redistribution registrations

- Attribute registrations
- Synchronization completion

RIB request counters and results

RIB also maintains counters for requests that RIB sends to its clients.

When RIB next-hop dampening is disabled, RIB immediately notifies a client when a registered next hop becomes resolved or unresolved.

RIB also maintains the results of these requests.

RIB quarantining

Explains how RIB detects mutually recursive routes, quarantines unstable paths, and reevaluates them until they are safe to install.

RIB quarantining is a protection mechanism that

- detects problematic route oscillation between routing protocols and RIB
- quarantines routes that create mutual recursion, and
- reduces CPU impact while the routing condition is being resolved.

Why RIB quarantining is needed

RIB quarantining addresses a persistent oscillation between routing protocols and RIB. This oscillation occurs when a route is repeatedly inserted into and withdrawn from RIB.

If the oscillation is not dampened, both the protocol process and the RIB process can experience high CPU use. This condition can affect the rest of the system and can block other protocol and RIB operations.

This behavior usually occurs when a specific combination of routes is received and installed in RIB. It typically results from a network misconfiguration that spans the network and cannot be detected on a single router during configuration.

How RIB quarantining works

The quarantining mechanism detects mutually recursive routes and quarantines the last route that completes the mutual recursion.

RIB periodically reevaluates the quarantined route to determine whether the mutual recursion still exists. If the recursion remains, the route stays quarantined. If the recursion is gone, the route is released from quarantine.

What happens during quarantine

1. RIB detects when a problematic path is installed.
2. RIB notifies the protocol that installed the path.
3. The protocol marks the route as quarantined. If the route is a Border Gateway Protocol (BGP) route, BGP does not advertise reachability for that route to its neighbors.
4. RIB periodically tests quarantined paths to determine whether they can be safely installed.
5. RIB notifies the protocol when the path is safe to use again.

Route consistency checker

Describes how RCC compares control-plane and data-plane routes to detect inconsistencies and support on-demand or background troubleshooting.

Route Consistency Checker (RCC) is a command-line tool that

- compares Routing Information Base (RIB) entries with Forwarding Information Base (FIB) entries
- detects inconsistencies between the control plane and the data plane, and
- provides details that help diagnose forwarding problems and traffic loss.

Why RCC is used

Routers in production networks can reach a state where the forwarding information does not match the control-plane information.

Possible causes include fabric or transport failures between the route processor (RP) and the line cards (LCs), or issues with the Forwarding Information Base (FIB).

RCC identifies these inconsistencies and provides detailed information about them. You can use this information to investigate the cause of forwarding problems and traffic loss.

How RCC works

RCC compares the Routing Information Base (RIB) against the Forwarding Information Base (FIB).

When RCC detects an inconsistency, the output identifies the specific route, the type of inconsistency, and additional data that supports troubleshooting.

RCC runs on the route processor. The FIB checks for errors on the line card and forwards the first 20 error reports to RCC.

RCC receives error reports from all nodes, checks them for exact matches, and adds them to either the soft queue or the hard queue.

Each queue can store up to 1000 error reports, and the queue does not prioritize entries. If different nodes report the same error, RCC logs the exact-match errors as one error.

RCC compares errors by using attributes such as prefix, version number, and error type.

RCC operating modes

RCC can run in two modes: on-demand scan and background scan.

On-demand scan

In on-demand scan mode, the user requests a scan from the command-line interface for a specific prefix in a specific table, or for all prefixes in a table.

RCC runs the scan immediately and publishes the results when the scan completes.

RCC performs on-demand scans per VRF.

Background scan

In background scan mode, the user configures a scan to run periodically during normal router operation.

The configuration specifies the interval for the periodic scan. You can configure this scan for a single table or for multiple tables.

RCC performs background scans for the default VRF and for non-default VRFs.

Enable RCC scans

Shows how to configure RCC background scanning and run on-demand scans to verify route consistency in IPv4 or IPv6 routing tables.

1. Configure an RCC background scan for the IPv6 unicast table.

   This example enables an RCC background scan with a period of 500 milliseconds between scan buffers for IPv6 unicast tables.

   ```
   Router(config)# rcc ipv6 unicast period 500
   ```

2. Run an RCC on-demand scan for a specific IPv4 prefix in a VRF.

   This example runs an RCC on-demand scan for prefix `10.10.0.0/16` in VRF `vrf1`.

   ```
   Router# show rcc ipv4 unicast 10.10.0.0/16 vrf vrf1
   ```

RCC runs periodic background scans or performs an immediate on-demand scan, depending on the option that you use.

BGP-RIB feedback mechanism for update generation

Explains how BGP waits for RIB and FIB installation feedback before advertising routes, helping prevent premature updates and traffic loss.

The BGP-RIB feedback mechanism for update generation is a synchronization mechanism that

- prevents premature route advertisements
- ensures that routes are installed locally before BGP advertises them to neighbors, and
- reduces packet loss that can occur when traffic is attracted before the data plane is programmed.

How the mechanism works

BGP waits for feedback from RIB to confirm that routes installed by BGP in RIB are also installed in the Forwarding Information Base (FIB) before BGP sends updates to neighbors.

RIB uses the Bulk Content Downloader (BCDL) feedback mechanism to determine which route versions FIB has consumed. RIB then updates BGP with that version information.

BGP sends updates only for routes whose versions are at or below the version that FIB has installed.

Why the mechanism is used

This selective update behavior prevents BGP from sending premature updates that can attract traffic before the data plane is programmed.

This protection is especially useful after a router reload, a line card online insertion and removal (OIR) event, or a link flap that makes an alternate path available.

Configure BGP to wait for RIB feedback before sending updates

Teaches you how to enable **update wait-install** so BGP advertises routes only after RIB confirms they are installed in FIB.

The **update wait-install** command delays BGP updates until RIB confirms that the routes are installed in FIB.

You can configure this command in router address-family IPv4 or router address-family VPNv4 configuration mode.

- Ensure that BGP is configured on the router.

- Ensure that you know whether you want to configure the setting under the IPv4 or VPNv4 address family.

1. Enter BGP address-family configuration mode.

```
Router# configure
Router(config)# router bgp 65000
Router(config-bgp)# address-family ipv4 unicast
```

   Use the address family that matches your deployment. For example, you can use IPv4 unicast or VPNv4 unicast address-family configuration mode.

2. Configure BGP to wait for RIB feedback before sending updates to neighbors.

```
Router(config-bgp-af)# update wait-install
```

3. Commit the configuration.

```
Router(config-bgp-af)# commit
```

4. Verify the configuration using these commands:
   - **show bgp**
   - **show bgp neighbors**
   - **show bgp process performance-statistics**

   These commands display information about the **update wait-install** configuration.

BGP waits for RIB feedback that confirms route installation in FIB before BGP sends route updates to neighbors.

Deploy and monitor RIB

Explains how to use RIB show commands to verify route state, inspect route views, troubleshoot connectivity between RIB and its clients, and monitor routing behavior across the network.

RIB is not configured separately for the Cisco IOS XR system. RIB computes router connectivity to other nodes in the network based on input from the routing protocols.

You can use RIB to monitor and troubleshoot the connections between RIB and its clients. You can also use RIB monitoring to check routing connectivity between nodes in a network.

To deploy and monitor RIB, you must understand the following concepts:

Verify RIB configuration using the routing table

Teaches you how to verify RIB operation by using route summary and detailed routing table commands to confirm expected route information.

Use this task to verify RIB operation from the routing table output.

Use the **show route** commands to check summary and detailed information for the IPv4 or IPv6 unicast routing table.

- Ensure that you have EXEC access on the router.
- Ensure that RIB and the routing protocols are configured on the router.

1. Display the route summary for the routing table.

```
Router# show route summary
```

This command displays route summary information for the specified routing table.

The default table that is summarized is the IPv4 unicast routing table.

2. Display detailed route information for the routing table.

```
Router# show route ipv4 unicast
```

This command displays detailed route information for the specified routing table.

You can enter the command with an IP address or other optional filters to limit the output. If you enter the command without filters, it displays all routes from the default IPv4 unicast routing table, which can produce extensive output depending on the network configuration.

You can confirm that RIB is running on the route processor and that the routing table contains the expected summary and detailed route information.

Troubleshoot networking and routing problems

Verifies the operation of routes between nodes to identify networking and routing issues.

Use this task to verify route operation between nodes and investigate routing-table behavior in RIB.

The **show route** command family displays active, backup, connected, local, longer-prefix, best-local, and next-hop route information.

1. Display the current routes in RIB.

```
Router# show route ipv4 unicast 192.168.1.11/8
```

2. Display the backup routes in RIB.

```
Router# show route ipv4 unicast backup 192.168.1.11/8
```

3. Display the best local address for return traffic from a destination.

```
Router# show route ipv4 unicast best-local 192.168.1.11/8
```

This command displays the best-local address to use for return packets from the specified destination.

4. Display the connected routes in the routing table.

```
Router# show route ipv4 unicast connected
```

This command displays the current connected routes in the routing table.

5. Display the local routes in the routing table.

```
Router# show route ipv4 unicast local
```

This command displays local routes for receive entries in the routing table.

6. Display the routes in RIB that share a given prefix.

```
Router# show route ipv4 unicast longer-prefixes 192.168.1.11/8
```

This command displays the current routes in RIB that share a given number of bits with a specified network.

7. Display the next-hop gateway or host for a destination address.

```
Router# show route ipv4 unicast next-hop 192.168.1.34
```

You can review active, backup, local, connected, longer-prefix, and next-hop route information to troubleshoot networking and routing problems.

Disable RIB next-hop dampening

Teaches you how to disable RIB next-hop dampening for the IPv4 or IPv6 address family.

Use this task to disable RIB next-hop dampening.

You can disable next-hop dampening for the IPv4 or IPv6 address family in RIB configuration mode.

1. Enter RIB configuration mode.

```
Router# configure
Router(config)# router rib
```

2. Disable next-hop dampening for the IPv4 or IPv6 address family.
   a) Disable next-hop dampening for the IPv4 address family.

```
Router(config-rib)# address-family ipv4 next-hop dampening disable
Router(config-rib)# commit
```

   b) Disable next-hop dampening for the IPv6 address family.

```
Router(config-rib)# address-family ipv6 next-hop dampening disable
Router(config-rib)# commit
```

RIB next-hop dampening is disabled for the selected address family.

RIB monitoring examples

Lists the RIB show commands to verify route state, inspect route views, troubleshoot connectivity between RIB and its clients, and monitor routing behavior across the network.

Use these **show route** command examples to monitor RIB routing connectivity and troubleshoot interactions between RIB and its clients.

Output of show route command: Example

Shows how the show route command displays the routing table, including route sources, connected networks, local addresses, static routes, and the gateway of last resort.

This example shows sample output from the **show route** command when it is entered without an address.

```
Router# show route

Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
       U - per-user static route, o - ODR, L - local

Gateway of last resort is 172.23.54.1 to network 0.0.0.0

C    10.2.210.0/24 is directly connected, 1d21h, HundredGigE 0/1/0/0
```

```
L    10.2.210.221/32 is directly connected, 1d21h, HundredGigE 0/1/0/0
C    10.6.100.0/24 is directly connected, 1d21h, Loopback1
L    10.6.200.21/32 is directly connected, 1d21h, Loopback0
S    192.168.40.0/24 [1/0] via 172.20.16.6, 1d21h
```

Output of show route backup command: Example

Shows how the show route backup command displays backup routes associated with installed routes, including alternate path source, metric, and interface details.

This example shows sample output from the **show route backup** command.

```
Router# show route backup

Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
         D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
         N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
         E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
         i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
        ia - IS-IS inter area, su - IS-IS summary null, * - candidate default

         U - per-user static route, o - ODR, L - local
  S    172.73.51.0/24 is directly connected, 2d20h, FourHundredGigE 0/0/0/1
                 Backup  O E2 [110/1] via 10.12.12.2, FourHundredGigE 0/0/0/2
```

Output of show route best-local command: Example

Shows how the show route best-local command displays the best local route entry for a specific prefix, including source, distance, metric, and interface.

This example shows sample output from the **show route best-local 10.12.12.1** command.

```
Router# show route best-local 10.12.12.1

  Routing entry for 10.12.12.1/32
    Known via "local", distance 0, metric 0 (connected)
    Routing Descriptor Blocks
      10.12.12.1 directly connected, via FourHundredGigE 0/0/0/0
        Route metric is 0
```

Output of show route connected command: Example

Shows how the **show route connected** command displays directly connected networks and their associated interfaces.

This example shows sample output from the **show route connected** command.

```
Router# show route connected

  C    10.2.210.0/24 is directly connected, 1d21h, HundredGigE 0/1/0/0
  C    10.6.100.0/24 is directly connected, 1d21h, Loopback1
```

Output of show route local command: Example

Shows how the **show route local** command displays local host routes assigned to router interfaces.

This example shows sample output from the **show route local** command.

```
Router# show route local

  L    10.10.10.1/32 is directly connected, 00:14:36, Loopback0
  L    10.91.36.98/32 is directly connected, 00:14:32, HundredGigE 0/1/0/0
  L   172.22.12.1/32 is directly connected, 00:13:35, FourHundredGigE 0/1/0/0
```

```
  L    192.168.20.2/32 is directly connected, 00:13:27, FourHundredGigE 0/0/0/0

  L    10.254.254.1/32 is directly connected, 00:13:26, FourHundredGigE 0/1/0/0
```

Output of show route ipv4 longer-prefixes command: Example

Shows how the **show route ipv4 longer-prefixes** command lists more specific IPv4 routes contained within a broader prefix.

This example shows sample output from the **show route ipv4 longer-prefixes** command.

```
Router# show route ipv4 longer-prefixes 172.16.0.0/8

  Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
         O - OSPF, IA - OSPF inter area, N1 - OSPF NSSA external type 1
         N2 - OSPF NSSA external type 2, E1 - OSPF external type 1
         E2 - OSPF external type 2, E - EGP, i - ISIS, L1 - IS-IS level-1
         L2 - IS-IS level-2, ia - IS-IS inter area
         su - IS-IS summary null, * - candidate default
         U - per-user static route, o - ODR, L - local

  Gateway of last resort is 172.23.54.1 to network 0.0.0.0
  S    172.16.2.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.3.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.4.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.5.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.6.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.7.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.8.0/32 is directly connected, 00:00:24, Loopback0
  S    172.16.9.0/32 is directly connected, 00:00:24, Loopback0
```

Output of show route resolving-next-hop command: Example

Shows how the **show route resolving-next-hop** command identifies the route used to resolve a specified next-hop address..

This example shows sample output from the **show route resolving-next-hop** command.

```
Router# show route resolving-next-hop 10.0.0.1

Nexthop matches 10.0.0.1/31
  Known via "connected", distance 0, metric 0 (connected)
  Installed Oct 21 10:11:22.460 for 00:03:02
  Directly connected nexthops
    directly connected, via FourHundredGigE0/0/0/0
      Route metric is 0
```

3 Routing Information Protocol

Topics:

Explains how RIP operates in small autonomous systems, including route exchange, version support, route selection, timers, redistribution, policies, authentication, and RIPng configuration.

The Routing Information Protocol (RIP) is a distance vector interior gateway protocol that

- exchanges routing information within an autonomous system in a small network
- supports the standard implementation of RIPv2, and
- provides backward compatibility with RIPv1 as specified by RFC 2453.

Table 3: Feature History Table

Feature Name	Release Information	Feature Description
RIPv2	Release 7.4.1	This feature enables RIP as the IGP of your network. RIP broadcasts UDP data packets to exchange routing information in networks that are flat rather than hierarchical, reducing network complexity and network management time.

Understanding RIP

Explains the capabilities, limitations, metrics, version behavior, and common use cases of RIPv2 on Cisco IOS XR.

Provides an overview of RIPv2 capabilities, including support for CIDR, route summarization, VLSM, and multicast advertisements, while detailing its limitations and configuration requirements on Cisco IOS XR software.

Routing Information Protocol (RIP) is a distance-vector routing protocol that exchanges routing information within a small autonomous system.

- RIP Version 1 supports only contiguous blocks of hosts, subnets, or networks in a single route.
- RIP Version 2 supports Classless Inter-Domain Routing (CIDR).
- RIP Version 2 also supports route summarization, variable-length subnet masks (VLSMs), autonomous systems, redistribution, and multicast address `224.0.0.9` for RIP advertisements.

RIP route metric limits

RIP utilizes specific metrics and update intervals to maintain routing information.

- Hop count: RIP uses hop count to measure route cost.
  - A directly connected network has a metric of zero.
  - An unreachable network has a metric of 16.
  - This limited metric range makes RIP unsuitable for large networks.
- Update interval: Advertises routing information every 30 seconds by default.

RIPv2 enhancements over RIPv1

RIP version support on Cisco IOS XR

RIPv2 provides several enhancements over the legacy RIPv1 protocol to support modern network requirements.

- Route summarization: Allows aggregation of network routes.
- Variable-length subnet masks (VLSMs): Supports efficient IP address allocation.
- Autonomous systems and the use of redistribution: Enables integration with other routing domains.
- Multicast address 224.0.0.9: Used for efficient RIP advertisements.

Cisco IOS XR supports only RIP Version 2 (RIP v2), as specified in RFC 2453.

- By default, the software sends and receives only RIP v2 packets.
- You can configure an interface to send only Version 1 packets, receive only Version 1 packets, use only Version 2 packets, or use both version types.

> **Note**
>
> VRF does not allow configuration of a group applied directly under router RIP. You can configure a group only if it is applied globally or under a VRF.

When to use RIP

RIP is widely used because it is simple to deploy and operate. RIP is a good choice in these situations:

- When the network must interoperate with diverse network devices.
- When the network is small and requires little bandwidth, configuration effort, and management overhead.
- When the network must support legacy host systems.

How RIP selects and advertises routes

Describes how RIP uses hop count to evaluate routes, marks unreachable paths, stores learned updates, and advertises routing information at regular intervals.

RIP selects routes by using hop count as its metric and advertises routing updates at regular intervals.

These stages describe how RIP manages routing information.

1. Evaluate route cost: RIP assigns a hop count to each route, representing the number of routers that the route traverses.

   RIP determines the relative cost of each route.

2. Identify reachable and unreachable routes: RIP treats a directly connected network as metric zero and an unreachable network as metric 16.

   RIP identifies which routes are usable and which routes are unreachable.

3. Advertise routing updates: RIP advertises routing information updates every 30 seconds by default.

   Neighboring routers receive periodic route information.

4. Store new route information: When RIP discovers new updates from neighbor routers, it stores them in the routing table.

   The routing table reflects the latest route information that RIP has learned.

RIP selects routes by hop count and maintains route information through periodic advertisements and routing table updates.

How split horizon works in RIP

Describes how split horizon suppresses route advertisements on the interface where they were learned to reduce routing loops and unnecessary updates.

Split horizon helps prevent routing loops in RIP on broadcast-type IP networks. It suppresses route advertisements on the interface from which the route information originated.

These stages describe how split horizon functions within the RIP domain.

1. Receive route information: A router receives route information on an interface from another router in the RIP domain.

   The router identifies the interface on which it learned the route.

2. Suppress route advertisement on the source interface: Split horizon prevents the router from advertising that route out of the same interface on which it was learned.

   The router reduces the possibility of routing loops.

3. Optimize communication after a link failure: This behavior helps optimize communication among multiple routers, especially when a link fails.

   The network reduces unnecessary route propagation during recovery.

4. Handle interfaces with secondary IP addresses: If an interface has secondary IP addresses and split horizon is enabled, RIP might not source updates from every secondary address. The router sources one routing update for each network number unless split horizon is disabled.

   The router controls how it sources updates on interfaces that have multiple addresses.

RIP uses split horizon to reduce routing loops and control how route updates are advertised on broadcast-type IP networks.

Route timers for RIP

Describes how RIP timers control update frequency, route validity, suppression, flushing, and output delay to improve convergence and reduce disruption during routing changes.

You can adjust these timers to tune RIP performance and reduce network disruption during routing changes.

RIP supports these timer settings:

- Update rate: Specifies the interval, in seconds, between routing updates.
- Invalid interval: Specifies how long, in seconds, a route remains valid without an update.
- Suppression interval: Specifies how long, in seconds, RIP suppresses information about better paths.
- Flush interval: Specifies how long, in seconds, RIP keeps a route before removing it from the RIP topology table.
- Output delay: Specifies the delay, in seconds, between RIP update packets.

Use the **timers basic** command to configure the update rate, invalid interval, suppression interval, and flush interval. Use the **output-delay** command to configure the delay between RIP update packets.

Adjusting these timers can improve convergence and help reduce disruption when the network switches to a redundant path.

Route redistribution for RIP

Explains how route redistribution injects routes between RIP and other routing domains so boundary routers can provide connectivity across mixed protocol environments.

Use this information to understand how RIP exchanges routing information with other routing protocols.

Route redistribution lets a router inject routes from one routing protocol into another. Routers that exchange routes between different routing domains are boundary routers. Without redistribution, routers in a routing domain know only the routes that belong to that domain.

You might use route redistribution with RIP for these reasons:

- To advertise routes from other protocols into RIP, such as static, connected, OSPF, and BGP.
- To migrate from RIP to a new Interior Gateway Protocol (IGP) such as EIGRP.
- To continue using RIP on some routers that support legacy host systems while you upgrade other routers.
- To exchange routes in a mixed-vendor network environment.

Route redistribution lets different parts of a network use the routing protocol that best fits their needs. This flexibility can reduce cost and support a wider range of technical requirements.

A simple redistribution case uses the **redistribute static** command to advertise static routes into RIP. More complex redistribution designs can introduce routing loops, incompatible routing information, or inconsistent convergence times. When multiple routing protocols run on the same router, review route selection behavior and administrative distance carefully.

Default administrative distances for RIP

Lists default administrative distance values for routing protocols and explains how they help determine route preference and redistribution behavior.

Use this information to understand the default administrative distance values that apply when you redistribute routes with RIP.

Administrative distance measures how much the router trusts a source of IP routing information. Lower values indicate higher trust. Higher values indicate lower trust. A value of 255 means that the router does not trust the source and ignores the route.

When you use RIP with route redistribution, review the default administrative distances for other route sources so that you can assign an appropriate distance value.

Table 4: Default administrative distances of route sources

Routing protocols	Administrative distance value
Connected interface	0
Static route out an interface	0
Static route to next hop	1
EIGRP Summary Route	5
External BGP	20
Internal EIGRP	90
OSPF	110
IS-IS	115
RIP version 1 and 2	120
External EIGRP	170
Internal BGP	200
Unknown	255

Routing policy options for RIP

Describes how RIP route policies are structured as independent configuration objects and explains the formatting rules required for valid policy definitions.

Use this information to understand how RIP route policies are structured and formatted.

A route policy is a configuration object that consists of related statements and expressions. The policy begins with the **route-policy** keyword and ends with the **end-policy** keyword. The router evaluates the statements in the policy together, not as separate commands.

Each line in a route policy is a logical unit. Use these formatting rules when you define a route policy:

- Add at least one new line after the **then** , **else** , and **end-policy** keywords.
- Add a new line after the closing parenthesis of a parameter list.
- Add a new line after the name string in a reference to an AS path set, community set, extended community set, or prefix set.

- Add at least one new line before the definition of a route policy, AS path set, community set, extended community set, or prefix set.
- You can add one or more new lines after an action statement.
- You can add one or more new lines after a comma separator in a named AS path set, community set, extended community set, or prefix set.
- End each logical unit of a policy expression with a new line. Do not insert a new line inside a logical unit.

RIP keychain authentication

Explains how RIP uses Cisco IOS XR keychains to authenticate inbound and outbound interface traffic and describes the supported authentication modes and failure behavior.

Use this information to understand how RIP authenticates interface traffic by using the Cisco IOS XR keychain infrastructure.

RIP keychain authentication uses the Cisco IOS XR security keychain infrastructure to store and retrieve secret keys. The router uses these keys to authenticate inbound and outbound RIP traffic on an interface.

Keychain management lets devices share secret keys before they establish trust. Routing protocols and network management applications use this mechanism to improve security when they communicate with peers.

> **Note**
>
> The Cisco IOS XR system security component provides keychain management. For more information about keychain concepts, configuration tasks, examples, and commands, see the keychain management documentation in the System Security Configuration Guide and System Security Command Reference.

> **Note**
>
> A keychain has no effect unless an application uses it for authentication. When you configure a keychain in the IOS XR keychain database and apply it to a RIP interface, the router uses that keychain to authenticate incoming and outgoing RIP traffic on that interface.

If you do not configure an authentication keychain on a RIP interface, the router treats RIP traffic on that interface as authentic and does not apply authentication checks.

RIP supports these authentication modes:

- keyed message digest mode: Use the **authentication keychain keychain-name mode md5** command
- clear text mode: Use the **authentication keychain keychain-name mode text** command

If a RIP interface is configured with a keychain, but the keychain does not exist in the keychain database or is not configured with the MD5 cryptographic algorithm, the router drops all inbound RIP packets on that interface. The router sends outbound RIP packets without authentication data.

Inbound RIP traffic on an interface

Use this information to understand how RIP validates inbound packets on an interface that is configured with a keychain.

RIP verifies inbound packets by using these rules:

Table 5: Inbound RIP packet authentication results

If	Then
The keychain configured on the RIP interface does not exist in the keychain database.	The router drops the packet and logs a RIP component-level debug message with details about the authentication failure.
The keychain is not configured with the MD5 cryptographic algorithm.	The router drops the packet and logs a RIP component-level debug message with details about the authentication failure.
The Address Family Identifier of the first entry in the message is not `0xFFFF`.	The router treats the packet as unauthenticated, drops it, and logs a RIP component-level debug message with details about the authentication failure.
The MD5 digest in the authentication data is invalid.	The router drops the packet and logs a RIP component-level debug message with details about the authentication failure.
None of these conditions occurs.	The router forwards the packet for further processing.

Outbound RIP traffic on an interface

Use this information to understand how RIP handles outbound packets on an interface that is configured with a keychain.

RIP processes outbound packets by using these rules:

Table 6: Outbound RIP packet authentication results

If	Then
The keychain is configured with the MD5 cryptographic algorithm.	The remote router can authenticate the RIP packet if it is configured with the same keychain.
The keychain is not configured with the MD5 cryptographic algorithm.	RIP packets fail authentication at the remote router.

Configure RIP

Teaches you how to enable the RIP process, define neighbors, and configure interface packet send and receive behavior for RIP version handling.

This reference provides the necessary commands to enable the RIP routing process, configure neighbor routers, and define version-specific packet handling for RIP interfaces.

Note

To save configuration changes, you must commit changes when the system prompts you.

Although you can configure RIP before you configure an IP address, no RIP routing occurs until at least one IP address is configured.

Enable RIP

Teaches you how to start the RIP process, configure a neighbor, and set interface send and receive behavior for RIP packet versions.

Use this task to enable RIP routing and establish a RIP routing process.

You can configure RIP before you configure an IP address. However, RIP routing does not occur until at least one IP address is configured.

This example uses neighbor address 172.160.1.2 and interface GigabitEthernet 0/1/0/0. Replace these sample values with values from your environment.

1. Enter RIP configuration mode and configure a RIP neighbor.

```
Router# configure
Router(config)# router rip
Router(config-rip)# neighbor 172.160.1.2
```

2. Configure RIP to send Version 2 packets to the broadcast IP address.

   You can apply this command at the interface level or the global RIP configuration level.

```
Router(config-rip)# broadcast-for-v2
```

3. Configure the RIP interface to receive and send RIP Version 1 and Version 2 packets.

```
Router(config-rip)# interface GigabitEthernet 0/1/0/0
Router(config-rip-if)# receive version 1 2
Router(config-rip-if)# send version 1 2
Router(config-rip-if)# commit
```

   RIP is enabled, the neighbor is configured, and the interface is set to send and receive RIP Version 1 and Version 2 packets.

Basic RIP configuration example

This example shows two Gigabit Ethernet interfaces configured with RIP.

```
Router(config)# interface GigabitEthernet0/6/0/0
Router(config-if)# ipv4 address 172.16.0.1 255.255.255.0
Router(config-if)# exit
Router(config)# interface GigabitEthernet0/6/0/2
Router(config-if)# ipv4 address 172.16.2.12 255.255.255.0
Router(config-if)# exit
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# exit
Router(config-rip)# interface GigabitEthernet0/6/0/2
```

RIP configuration on a provider edge router example

This example shows how to configure basic RIP on the provider edge router with two VPN routing and forwarding (VRF) instances.

```
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# exit
Router(config-rip)# vrf vpn0
Router(config-rip-vrf)# interface GigabitEthernet0/6/0/2
Router(config-rip-if)# exit
Router(config-rip-vrf)# exit
Router(config-rip)# vrf vpn1
Router(config-rip-vrf)# interface GigabitEthernet0/6/0/3
```

Customize RIP behavior

Teaches you how to adjust RIP summarization, timers, delays, NSF, and interface update behavior to match network requirements.

Use this task to customize RIP for network timing and the acceptance of route entries.

This example uses interface `GigabitEthernet 0/1/0/0`. Replace the sample interface and timer values with values from your environment.

Each command in this task is optional. Configure only the settings that your network requires.

1. Enter RIP configuration mode.

```
Router# configure
Router(config)# router rip
```

2. Enable automatic route summarization.

   By default, `auto-summary` is disabled. If the network contains disconnected subnets, use the `no` form of this command so that the router can send subnet and host routing information across classful network boundaries.

```
Router(config-rip)# auto-summary
```

3. Configure RIP network timers.

   Use this command to adjust the RIP update, invalid, holddown, and flush timers. You can view the current and default timer values in the output of the `show rip` command.

```
Router(config-rip)# timers basic 5 15 15 30
```

4. Configure the delay between RIP update packets.

   Use this command when a high-end router sends updates to a lower-speed router that cannot receive packets at the default rate.

```
Router(config-rip)# output-delay 10
```

5. Enable nonstop forwarding (NSF) for RIP routes.

```
Router(config-rip)# nsf
```

6. Enter RIP interface configuration mode.

```
Router(config-rip)# interface GigabitEthernet 0/1/0/0
```

7. Allow the interface to accept route entries that are received with metric zero.

   The router changes a received metric of zero to a metric of one.

```
Router(config-rip-if)# metric-zero-accept
```

8. Disable split horizon on the interface.

   Split horizon is enabled by default. Change this setting only if the network requires a different advertisement behavior.

```
Router(config-rip-if)# split-horizon disable
```

9. Enable poison reverse processing for RIP updates and commit the configuration.

```
Router(config-rip-if)# poison-reverse
Router(config-rip-if)# commit
```

The router applies the configured RIP customization settings.

RIP uses the configured route summarization, timers, delay, NSF, and interface settings.

RIP timer configuration for each VRF example

This example shows how to adjust RIP timers for each VPN routing and forwarding (VRF) instance.

For VRF instance `vpn0`, the **timers basic** command sets the update, invalid, suppression, and flush timers to `10`, `30`, `30`, and `45` seconds.

For VRF instance `vpn1`, the **timers basic** command sets the timers to `20`, `60`, `60`, and `70` seconds. The **output-delay** command sets the delay between RIP update packets to `10` milliseconds.

```
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# exit
Router(config-rip)# vrf vpn0
Router(config-rip-vrf)# interface GigabitEthernet0/6/0/2
Router(config-rip-if)# exit
Router(config-rip-vrf)# timers basic 10 30 30 45
Router(config-rip-vrf)# exit
Router(config-rip)# vrf vpn1
Router(config-rip-vrf)# interface GigabitEthernet0/6/0/3
Router(config-rip-if)# exit
Router(config-rip-vrf)# timers basic 20 60 60 70
Router(config-rip-vrf)# output-delay 10
Router(config-rip-vrf)# commit
```

Configure default route origination for RIP

Teaches you how to advertise the default route to RIP peers and notes that routers configured for default origination ignore peer-advertised default routes.

Starting from IOS XR Release 7.5.2, if a router is configured with the **default-information originate** command, it ignores default routes that it receives from RIP peers. The router does not accept or install a peer-advertised default route in the RIP database or the RIB.

Configure RIP to advertise the default route `0.0.0.0/0` to RIP peers. A receiving router can install the default route in the RIP database and the routing information base (RIB).

Enter the RIP configuration mode and enable the default route advertisement.

```
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/2/0/0
Router(config-rip-if)# exit
Router(config-rip)# default-information originate
```

Control RIP routing updates

Shows how to control RIP update exchange and route propagation on selected interfaces and neighbors.

Use this task to control or prevent RIP routing update exchange and propagation.

You can control RIP updates to reduce update traffic on WAN links, help prevent routing loops, filter received route information, prevent dynamic route processing on an interface, and preserve bandwidth.

This example uses neighbor address `172.160.1.2` and interfaces `GigabitEthernet 0/1/0/0` and `GigabitEthernet 0/2/0/0`. Replace these sample values with values from your environment.

1. Enter RIP configuration mode, configure a RIP neighbor, and enter RIP interface configuration mode.

```
Router# configure
Router(config)# router rip
Router(config-rip)# neighbor 172.160.1.2
Router(config-rip)# interface GigabitEthernet 0/1/0/0
```

2. Suppress RIP updates on the interface.

   This command suppresses RIP updates on the interface, but the router can still send updates to explicitly configured neighbors.

```
Router(config-rip-if)# passive-interface
Router(config-rip-if)# exit
```

3. Apply a route policy to a RIP interface and commit the configuration.

   You can use a route policy to control the updates that the router advertises or receives on the interface.

```
Router(config-rip)# interface GigabitEthernet 0/2/0/0
Router(config-rip-if)# route-policy out
Router(config-rip-if)# commit
```

   The router controls RIP update exchange on the selected interfaces and neighbors.

RIP passive interface and explicit neighbor example

This example shows how to configure passive interfaces and explicit neighbors. When an interface is passive, it receives routing updates but does not send them unless the destination is an explicitly configured neighbor.

```
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# passive-interface
Router(config-rip-if)# exit
Router(config-rip)# interface GigabitEthernet0/6/0/2
Router(config-rip-if)# exit
Router(config-rip)# neighbor 172.17.0.1
Router(config-rip)# neighbor 172.18.0.5
```

RIP route control example

This example shows how to use the `distance` command to control the installation of RIP routes in the Routing Information Base (RIB) and how to use the `maximum-paths` command to control the number of parallel paths for each RIP route.

```
Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# route-policy polin in
Router(config-rip-if)# exit
Router(config-rip)# distance 110
Router(config-rip)# maximum-paths 8
```

Create route policies for RIP

Teaches you how to define a route policy, set RIP route attributes such as metric, and apply the policy to RIP updates.

A route policy starts with the **route-policy** command, policy statements are added to the route policy, and it ends with the **end-policy** command. A route policy is useful only when it is applied to a routing protocol's routes.

1. Create the route policy.

```
Router# configure
Router(config)# route-policy IN-IPv4
```

2. Set the RIP metric attribute and commit the configuration.

   This example sets the RIP metric to `42` for routes that match the policy.

```
Router(config-rpl)# set rip metric 42
Router(config-rpl)# end-policy
Router(config-rpl)# commit
```

3. Apply the routing policy to updates advertised to or received from an RIP neighbor.

   Use the route policy direction that matches your requirement.

   Table 7: Route policy direction

If...	Then...
You want to filter or modify received RIP updates.	Apply the policy with the `in` keyword.
You want to filter or modify advertised RIP updates.	Apply the policy with the `out` keyword.

```
Router# configure
Router(config)# router rip
Router(config-rip)# route-policy IN-IPv4 in
Router(config-rip)# commit
```

Redistribute routes into RIP

This example shows how to redistribute Border Gateway Protocol (BGP) and static routes into RIP.

The RIP metric for redistributed routes is determined by the route policy. If no route policy is configured, or if the route policy does not set the RIP metric, the metric is derived from the redistributed protocol. For VPNv4 routes redistributed by BGP, the router uses the RIP metric set on the remote provider edge (PE) router when that metric is valid.

For BGP, IS-IS, OSPF, EIGRP, connected, and static routes in other cases, the router uses the metric configured by the **default-metric** command. If the router cannot determine a valid metric, redistribution does not occur.

```
Router(config)# route-policy ripred
Router(config-rpl)# set rip metric 5
Router(config-rpl)# end-policy

Router(config)# router rip vrf vpn0
Router(config-rip)# interface GigabitEthernet0/6/0/2
Router(config-rip)# redistribute connected default-metric 3
Router(config-rip)# vrf vpn1
Router(config-rip-vrf)# interface GigabitEthernet0/6/0/3
Router(config-rip-vrf)# redistribute bgp 100 route-policy ripred
Router(config-rip-vrf)# redistribute static
Router(config-rip-vrf)# default-metric 3
```

Configure inbound and outbound route policies for RIP

This example shows how to configure inbound and outbound route policies to control which route updates a RIP interface receives and sends.

```
Router(config)# prefix-set pf1 10.1.0.0/24
Router(config-pfx)# end-set

Router(config)# prefix-set pf2 198.51.100.0/24
Router(config-pfx)# end-set

Router(config)# route-policy policy_in
Router(config-rpl)# if destination in pf1 then
Router(config-rpl-if)# pass
Router(config-rpl-if)# endif
Router(config-rpl)# end-policy

Router(config)# route-policy pass-all
Router(config-rpl)# pass
Router(config-rpl)# end-policy

Router(config)# route-policy infil
Router(config-rpl)# if destination in pf2 then
Router(config-rpl-if)# add rip metric 2
Router(config-rpl-if)# pass
Router(config-rpl-if)# endif
Router(config-rpl)# end-policy

Router(config)# router rip
Router(config-rip)# interface GigabitEthernet0/6/0/0
Router(config-rip-if)# route-policy policy_in in
Router(config-rip)# interface GigabitEthernet0/6/0/2
Router(config-rip-if)# route-policy infil in
Router(config-rip-if)# route-policy pass-all out
```

Configure RIP authentication keychain

Teaches you how to apply MD5 or clear-text keychain authentication to RIP interfaces in default or non-default VRFs.

Use this task to configure keyed message digest 5 (MD5) or clear-text authentication for RIP on an interface.

Configure all keychains in the Cisco IOS XR keychain database before you apply them to a RIP interface or VRF.

The **authentication keychain** and **mode md5** commands accept a keychain name that is not yet configured in the IOS XR keychain database, or a keychain that does not use the MD5 cryptographic algorithm. In either case, the router drops all incoming packets on the interface and sends outgoing packets without authentication data.

- Configure the required keychain in the Cisco IOS XR keychain database.
- Review the keychain configuration commands in the *Implementing Keychain Management* module of the *System Security Configuration Guide*.
- Ensure that RIP is enabled on the router.

1. Enter RIP configuration mode.

```
Router# configure
Router(config)# router rip
```

2. If you are configuring a non-default VRF, enter the VRF and interface configuration mode.

```
Router(config-rip)# vrf vrf_rip_auth
Router(config-rip-vrf)# interface POS 0/6/0/0
```

   Skip this step if you are configuring the default VRF.

3. If you are configuring the default VRF, enter the interface configuration mode.

```
Router(config-rip)# interface POS 0/6/0/0
```

   Use this step only for the default VRF.

4. Apply the authentication keychain to the interface.

```
Router(config-rip-if)# authentication keychain key1 mode md5
```

   - `authentication keychain key1 mode md5`
   - `authentication keychain key1 mode text`

   Use `md5` for keyed message digest authentication or `text` for clear-text authentication.

5. Commit the configuration.

```
Router(config-rip-if)# commit
```

The RIP interface uses the configured keychain for authentication in the selected VRF.

Apply a RIP authentication keychain on a default VRF interface

```
Router(config)# router rip
Router(config-rip)# interface POS0/6/0/0
Router(config-rip-if)# authentication keychain key1 mode md5
Router(config-rip-if)# end
```

Apply a RIP authentication keychain on a non-default VRF interface

```
Router(config)# router rip
Router(config-rip)# vrf rip_keychain_vrf
Router(config-rip-vrf)# interface POS0/6/0/0
Router(config-rip-if)# authentication keychain key1 mode md5
Router(config-rip-if)# end
```

Routing Information Protocol next generation

Explains how RIPng extends RIP to IPv6 by supporting IPv6 route exchange, FF02::9 multicast updates, route filtering, default origination, and IPv6 routing behavior similar to IPv4 RIP.

Routing Information Protocol next generation (RIPng) is the IPv6 implementation of RIP that

- supports IPv6 addresses and prefixes for route exchange
- uses the all-RIP-devices multicast group address `FF02::9` for RIP update messages, and
- provides IPv6 routing behavior that is functionally similar to RIP for IPv4.

Feature History Table

Feature Name	Release Information	Feature Description
Routing Information Protocol next generation (RIPng)	Release 7.5.2	This feature enables RIP enhancements for IPv6 that include support for IPv6 addresses and prefixes, and the use of the all-RIP-devices multicast group address FF02::9 as the destination address for RIP update messages. This feature functions the same and offers the same benefits as RIP in IPv4.

How RIPng stores routes

Based on RFC 2080, each RIPng process maintains a local routing table called the Routing Information Database (RIB).

The RIPng RIB contains the best-cost IPv6 RIP routes learned from neighboring devices. When RIPng learns the same route from different neighbors with different costs, it keeps only the lowest-cost route in the local RIB.

The RIPng RIB also stores expired routes that the RIP process continues to advertise to neighboring devices that run RIPng.

RIPng attempts to install every nonexpired route from its local RIB into the primary IPv6 RIB. If the router learns the same route from another routing protocol with a better administrative distance, the RIPng route is not added to the primary IPv6 RIB. The route remains in the RIPng RIB.

RIPng functional capabilities

- Route filtering on IPv6 RIP interfaces and VRFs by using route policies
- Default-information origination and generation for IPv6 RIP
- Poison-reverse and split-horizon for IPv6 RIP
- Configuration of administrative distance

Configure RIPng on an interface in a VRF

Teaches you how to enable RIPng for IPv6 in a VRF, associate an interface with the RIP process, and verify interface, process, statistics, and database status.

Use this task to enable RIPng on an interface in a VRF and verify that the RIPng process is active for IPv6 routing updates.

This task configures RIPng under a VRF by enabling the IPv6 address family and associating an interface with the RIP process.

- Ensure that the VRF is configured on the router.
- Ensure that IPv6 is enabled on the interface that you want to use for RIPng.

1. Enter RIP configuration mode and select the VRF.

```
Router(config)# router rip
Router(config-rip)# vrf vrf1
```

2. Enable the IPv6 address family for the VRF.

```
Router(config-rip-vrf)# address-family ipv6
```

3. Associate the interface with RIPng.

```
Router(config-rip-vrf-af)# interface GigabitEthernet 0/0/0/0
Router(config-rip-vrf-af)# commit
```

RIPng is enabled for IPv6 on the specified interface in the selected VRF.

4. Verify RIPng IPv6 VRF interface information.

```
Router# show rip ipv6 vrf vrf1 interface
Fri Dec  9 17:39:05.855 IST

GigabitEthernet0/0/0/0
Rip enabled?:                Yes
Out-of-memory state:         Normal
Accept Metric 0:             No
Interface state:             Up
IP address:                  10:10:10::2/64
Metric Cost:                 0
Split horizon:               Enabled
Poison Reverse:              Disabled
Socket set options:
    Joined multicast group:    Yes
    LPTS filter set:           Yes

  Total packets received: 0
```

5. Verify RIPng IPv6 process-level, VRF, Routing Descriptor Block (RDB), and interface information.

```
Router# show rip ipv6 vrf vrf1 internal

Fri Dec  9 17:45:02.219 IST

RIP process level information
-----------------------------
Socket descriptor                       63
UDP connected:                         Yes
OOM state:                              1
OOM Severe state time:                  60 seconds
OOM Severer state timer running:       No
Number of routes alloced:               1
Number of paths alloced:                1
Garbage collection time:                300 seconds
RIB Batch buffer: Max buf length:       1434700
                  Max msg length:       524256
                  Max data length:      0
                  Msg length:           0
Number of RIB transit routes:           0
RIB RDB Q size:                         0

RIP VRF information
-------------------
Name:                                   vrf1
ID:                                     0x60000002
NSF:                                    No
NSF Lifetime:                           0
Distance:                               120
DistanceIP Q size:                      0
Active:                                 Yes
Added to socket:                        Yes
PSL Send Q size:                        0
```

```
OOM Flags:                                0x0
Number of routes alloced:                 1
Number of paths alloced:                  1

RIP RDB information
-------------------
Table ID:                                 0xe0800011
Q flags:                                  0x0
Active:                                   Yes
Handle:                                   0x2
Old Handle:                               0xffff
Connected Handle:                         0x0
BGP Handle:                               0xffff
Table Ready:                              Yes
Default Info originate:                   No
RedInfo Q size:                           0
RIB Update Q size:                        0
RIB Delete pending:                       No
Triggered Update Q size:                  0
Triggered Update batch count:             0
Garbage Q size:                           0
Converged:                                Yes
Convergence timer expired:                Yes
Convergence timer running:                No
Ageout timer running:                     Yes
Garbage timer running:                    Yes (231 seconds left)
OM Triggered Update timer running:        No
WorkQueue Request Q size:                 0

RIP Interface information
-------------------------
Cumulative Peer Q size:                   0
```

6. Verify RIPng IPv6 statistics.

```
Router# show rip ipv6 vrf vrf1 statistics
Fri Dec  9 17:45:12.223 IST

RIPv6 statistics:
Total messages sent:        1
Message send failures:      0
Regular updates sent:       0
Queries responsed to:       0
RIB updates:                0
Total packets received:     0
Discarded packets:          0
Discarded routes:           0
Packet received at standby: 0
Number of routes allocated: 1
Number of paths allocated:  1
Route malloc failures:      0
Path malloc failures:       0
```

7. Verify the RIPng IPv6 database and interface information

```
Router# show rip ipv6 vrf vrf1 database
Fri Dec  9 17:44:14.868 IST

Routes held in RIP's topology database:
```

```
10:10:10::/64
    [0]    directly connected, GigabitEthernet0/0/0/0
```

```
RP/0/0/CPU0:PE2# show rip ipv6 vrf vrf1 interface
Fri Dec  9 17:44:20.547 IST

GigabitEthernet0/0/0/0
Rip enabled?:               Yes
Out-of-memory state:        Normal
Accept Metric 0:            No
Interface state:            Up
IP address:                 10:10:10::2/64
Metric Cost:                0
Split horizon:              Enabled
Poison Reverse:             Disabled
Socket set options:
    Joined multicast group:    Yes
    LPTS filter set:           Yes

  Total packets received: 0
```

4 Routing Policy Language

Topics:

Explains how Routing Policy Language supports modular policy creation, reusable components, route matching, attribute modification, validation, execution behavior, and aggregation control for routing policies.

Routing policy language

Explains how RPL enables reusable policy blocks, route attribute matching, and explicit control over route acceptance, rejection, and modification for routing protocols such as BGP, IS-IS, and OSPF.

A routing policy language (RPL) is a configuration language that

- enables modular, reusable policy blocks for configuration management
- supports matching and modifying route attributes for protocols such as BGP, IS-IS, and OSPF, and
- allows explicit control over route acceptance, rejection, or modification within policy definitions.

Routing policy language for modular policy design
Table 8: Feature History Table

Feature Name	Release Information	Feature Description
Routing Policy Language	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.
Routing Policy Language	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) This feature simplifies large-scale routing configurations by enabling modular policy building, reducing the complexity and maintenance of configuration information. RPL enables policy creation using independently defined, parameterized blocks that can be reused across different policies, reducing redundancy. *This feature is supported on: - 8212-48FH-M - 8711-32FH-M - 8712-MOD-M - 88-LC1-36EH - 88-LC1-12TH24FH-E - 88-LC1-52Y8H-EM

Routing Policy Language (RPL) is designed to support large-scale routing configurations through a modular and structured policy framework. Unlike traditional route maps, access lists, and prefix lists, RPL enables policies to be built from reusable, independently maintained components. These components can be combined to form complete policies, reducing configuration duplication and simplifying maintenance.

RPL also supports parameterization, allowing policies that share the same structure but differ in specific values to be implemented as a single reusable policy with variable inputs. This approach improves scalability and consistency across deployments.

The language introduces the concept of sets, which are containers used for route attribute matching and setting operations. RPL supports prefix sets, community sets, AS path sets, and extended community sets. Sets group related routing data such as prefixes, community values, or AS path expressions. For small value groupings, inline sets can be defined directly within a policy, eliminating the need for separately named sets.

Policy decisions, such as accepting or rejecting routes, are explicitly defined within the policy. RPL uses Boolean logic (AND, OR, NOT) to construct conditional expressions based on matching operations. These conditions are evaluated using structured control flow statements such as `if`, `elseif`, and `else`, allowing precise definition of policy behavior.

Routing policy language structures

Describes the structure of RPL objects and sets, including named and inline forms, matching constructs, policy statements, conditional logic, apply behavior, parameterization, and supported set types used to build routing policies.

A routing policy language structure is a set of rules and syntax that

- organizes how routing policies and sets are defined and named
- specifies the valid elements and formats for sets such as as-path-sets, community-sets, extcommunity-sets, prefix-sets, and rd-sets, and
- enables you to create, match, and manage routing policies effectively.

Routing policy language structures allow persistent, named objects to be created for both policies and sets. These sets serve as containers for matching various attributes, such as AS paths or communities, in policy conditions. Both inline and named forms are supported, providing flexibility for different configuration needs.

Policy objects in RPL

Describes policy objects as named, persistent RPL definitions and outlines how policies and sets are identified, structured, and managed within routing policy configurations.

A policy object is a persistent, named configuration item in the Routing Policy Language (RPL) that can be defined as a set or a policy and is identified by a unique name.

To define a policy named `test`, use this structure.

```
route-policy test
[ . . . policy statements . . . ]
end-policy
```

Policy names can include uppercase and lowercase letters, numerals (0-9), period (.), hyphen (-), and underscore (_), and must begin with a letter or number.

Sets in RPL

Explains how sets group unique values for matching operations, compares named and inline forms, and outlines evaluation behavior for populated and empty sets in policy conditions.

A set is an unordered collection of unique elements in RPL that

- is used as a container for groups of values for matching in conditional expressions
- allows both named and inline forms, and
- separates elements with commas, allowing null (empty) sets except where prohibited.
- Named set: Defined and referenced by name.
- Inline set: Defined inline within a policy for quick comparisons.

Inline sets

Example: Inline set

Example: Equivalent named set

Empty sets and evaluation behavior

In this context, the term `set` is used in its mathematical sense to refer to an unordered collection of unique elements. In the routing policy language, sets serve as containers for grouped values that are used in matching operations within conditional expressions.

Elements in a set are separated by commas, and empty (null) sets are permitted.

When comparisons involve only a small number of elements, such as two or three community values, the values can be listed directly within the policy. These are referred to as inline sets.

Inline sets are functionally equivalent to named sets but allow simple comparisons without defining a separate set. This approach avoids maintaining a standalone set when only a small number of values are required.

In general, the syntax for an inline set is a comma-separated list enclosed in parentheses, where each entry corresponds to the required data type, for example, a prefix or community value.

```
route-policy sample-inline
  if community matches-any ([10..15]:100) then
    set local-preference 100
  endif
end-policy
```

```
community-set test-communities
  10:100,
  11:100,
  12:100,
  13:100,
  14:100,
  15:100
end-set

route-policy sample
  if community matches-any test-communities then
    set local-preference 100
  endif
end-policy
```

Both policies produce the same result. The inline form eliminates the need to configure a separate set when only a limited number of values are being compared. Either approach can be used depending on the configuration context.

Subsequent sections provide examples of both named and inline set usage where appropriate.

Consider this example:

```
prefix-set backup-routes
  # currently no backup routes are defined
end-set
```

If this set is referenced in a condition, `if destination in backup-routes then`, the condition evaluates to FALSE for every route because the set contains no matching entries.

AS path sets

Describes how AS path sets use regular expressions in named or inline form to match route AS path attributes and notes performance considerations for regex-based evaluations.

An AS path set is a collection of match criteria for Autonomous System (AS) path attributes that

- supports matching using regular expressions,
- can be defined as a named set or an inline set, and
- is used to match AS path attributes in route policies.

Named AS path set

Inline AS path set

Performance considerations

The named AS path set form uses the **ios-regex** keyword to specify the regular expression type. Enclose each regular expression in single quotation marks.

```
as-path-set aset1
  ios-regex '_42$',
  ios-regex '_127$'
end-set
```

This AS path set contains two elements. In a matching operation, the system matches any route whose AS path ends with autonomous system (AS) number 42 or 127.

To remove this named AS path set, use the **no as-path-set aset1** command.

An inline AS path set is defined as a list enclosed in parentheses and separated by commas.

```
(ios-regex '_42$', ios-regex '_127$')
```

This inline set matches the same AS paths as the named set, aset1, but does not require you to create a separate named set. It can be used directly within a policy where needed.

Regular expression matching requires significant CPU resources. You can improve policy performance by:

- collapsing multiple regular expression patterns to reduce the total number of regular expression evaluations, or
- using equivalent native AS path match operations such as **as-path neighbor-is**, **as-path originates-from**, or **as-path passes-through**.

Community sets

Explains how community sets store BGP community values in named or inline form, support parameterization and wildcards, and enforce validation rules for usable policy matching.

A community set is a named or inline set in RPL that

- holds BGP community values for matching against the community attribute
- requires community values as two unsigned decimal integers (0 to 65535) separated by a colon, and
- supports parameterization and wildcards.

Named community set

Inline community set

Well-known community values

Wildcard support

Validation rules

A named community set is a user-defined collection of community values. It is configured with a unique name and referenced within routing policies to match the BGP community attribute.

Example:

```
community-set cset1
  12:34,
  12:56,
  12:78,
  internet
end-set
```

Define an inline community set as a list enclosed in parentheses and separated by commas. Use this list directly within a policy when needed.

The inline form also supports parameterization. Each 16-bit portion of the community value can be parameterized.

Example:

```
(12:34, 12:56, 12:78)
($as:34, $as:$tag1, 12:78, internet)
```

RPL provides symbolic names for standard well-known community values:

- **internet** represents 0:0
- **no-export** represents 65535:65281
- **no-advertise** represents 65535:65282, and
- **local-as** represents 65535:65283.

Community specifications may include wildcards. Define a wildcard by replacing one of the 16-bit portions of the community value with an asterisk. The wildcard matches any value for that portion.

Example:

```
community-set cset3
  123:*
end-set
```

This set matches all communities in which the autonomous system portion is 123.

Every community set must contain at least one community value. If a community set is empty, the system rejects it as invalid.

Extended community sets
Describes extended community sets, including rt and soo forms, and explains how named and inline definitions support parameterization, wildcards, and policy matching requirements.

An extended community set is a collection of extended-community values used in routing policies that

- supports both named and inline forms, including parameterization within parameterized policies
- permits matching using wildcards and regular expressions, and
- contains at least one extended community value.

Extended community rt set

Extended community soo set

An extended community set is similar to a community set, but it contains extended community values instead of regular community values. Extended community sets support both named and inline forms.

As with community sets, the inline form supports parameterization within parameterized policies. Either portion of the extended community value can be parameterized.

Community specifications may include wildcards. Define a wildcard by replacing one of the 16-bit portions of the community value with an asterisk. The wildcard matches any value for that portion.

Every extended community set must contain at least one community value. If a community set is empty, the system rejects it as invalid.

There are three types of extended community sets:

- cost
- soo
- rt

An rt set is an extended community set used to store BGP Route Target (RT) extended community type communities.

Example: Named form for extended community rt set

```
extcommunity-set rt a_rt_set
  203.0.113.4:666
  1234:666,
  203.0.113.4:777,
  4567:777
end-set
```

Example: Inline set form for extended community rt set

```
(203.0.113.4:666, 1234:666, 203.0.113.4:777, 4567:777)
($ipadrr:666, 1234:$tag, 1.2.3.4:777, $tag2:777)
```

A soo set is an extended community set used to store BGP site-of-origin (soo) extended community type communities.

Example: Named form for extended community soo set

```
extcommunity-set soo a_soo_set
  192.0.2.1:100,
  100:200
end-set
```

This table lists the options you can define under **extcommunity-set**.

Table 9: Supported options under extcommuniety-set rt and extcommunity-set soo

Option	Description
#-remark	Remark beginning with '#'

Option	Description
*	Wildcard (any community or part thereof)
<1-4294967295>	32-bit decimal number
<1-65535>	16-bit decimal number
A.B.C.D/M:N	Extended community - IPv4 prefix format
A.B.C.D:N	Extended community - IPv4 format
ASN:N	Extended community - ASPLAIN format
X.Y:N	Extended community - ASDOT format
abort	Discard RPL definition and return to top level config
dfa-regex	DFA style regular expression
end-set	End of set definition
exit	Exit from this submode
ios-regex	Traditional IOS style regular expression
show	Show partial RPL configuration

Prefix sets

Explains how prefix sets define IPv4 and IPv6 match criteria, including mask, ge, le, and eq options, validation rules, examples, and ACL-style wildcard matching support.

A prefix set is a set in RPL that

- holds IPv4 or IPv6 prefix match specifications
- allows specification of address, mask length, minimum and maximum matching length, and
- uses a comma-separated list of prefix specifications.

A prefix set contains IPv4 or IPv6 prefix match specifications. Each specification consists of four components:

- address
- mask length
- minimum matching length, and
- maximum matching length.

The address is mandatory. The remaining components are optional.

Prefix match specification format

Table 10: Prefix match specification components and defaults

Component	Description	IPv4 Range / Default	IPv6 Range / Default
Address	Expressed as a standard dotted-decimal IPv4 address or colon-separated hexadecimal IPv6 address.	Standard IPv4 format	Standard IPv6 format

Component	Description	IPv4 Range / Default	IPv6 Range / Default
Mask length	• Optional • Follows the address, separated by a slash (/) • Must be a nonnegative decimal integer	• Range: 0–32 • Default (if not specified): 32	• Range: 0–128 • Default (if not specified): 128
Minimum matching length (ge)	• Optional. • Specified using the *ge* keyword (greater than or equal to) followed by a nonnegative decimal integer.	• Range: 0–32 • Default: mask length	• Range: 0–128 • Default: mask length

Maximum matching length (le)	• Optional.
	• Specified using the *le* keyword (less than or equal to) followed by a nonnegative decimal integer.
		• Range: 0–32
		• Default rules:
		  • If *ge* is specified and *le* is not: 32
		  • If neither *ge* nor *le* is specified: mask length
			• Range: 0–128
			• Default rules:
			  • If *ge* is specified and *le* is not: 128
			  • If neither *ge* nor *le* is specified: mask length

Exact match (eq)	Syntactic shortcut to specify an exact prefix length for matching.	Matches only the specified prefix length	Matches only the specified prefix length

Prefix set structure and examples

A prefix set is defined as a comma-separated list of prefix match specifications.

IPv4 examples

```
prefix-set legal-ipv4-prefix-examples
  192.0.2.1,
  192.0.2.2/24,
  192.0.2.3/24 ge 28,
  192.0.2.4/24 le 28,
  192.0.2.5/24 ge 26 le 30,
  192.0.2.6/24 eq 28,
  192.0.2.7/32 ge 16 le 24,
  192.0.2.8/26 ge 8 le 16
end-set
```

Explanation of IPv4 examples:

- `192.0.2.1` matches only 192.0.2.1/32.
- `192.0.2.2/24` matches only 192.0.2.2/24.
- `192.0.2.3/24 ge 28` matches prefixes from 192.0.2.3/28 to 192.0.2.255/32.

- `192.0.2.4/24 le 28` matches prefixes from 192.0.2.4/24 to 192.0.2.240/28.
- `192.0.2.5/24 ge 26 le 30` matches prefixes from 192.0.2.5/26 to 192.0.2.252/30.
- `192.0.2.6/24 eq 28` matches prefixes of length 28 from 192.0.2.0/28 through 192.0.2.240/28.
- `192.0.2.7/32 ge 16 le 24` matches prefixes of length 32 in the range 192.0.[0..255].7/32 (from 192.0.0.7/32 to 192.0.255.7/32).
- `192.0.2.8/26 ge 8 le 16` matches prefixes of length 26 in the range 192.[0..255].2.8/26 (from 192.0.2.8/26 to 192.255.2.8/26).

IPv6 examples

```
prefix-set legal-ipv6-prefix-examples
  2001:DB8:1::/64,
  2001:DB8:2::/64 ge 96,
  2001:DB8:2::/64 ge 96 le 100,
  2001:DB8:2::/64 eq 100
end-set
```

Explanation of IPv6 examples:

- `2001:DB8:1::/64` matches only the single prefix 2001:DB8:1::/64.
- `2001:DB8:2::/64 ge 96` matches prefixes derived from 2001:DB8:2::/64 with lengths from /96 through /128.
- `2001:DB8:2::/64 ge 96 le 100` matches prefixes derived from 2001:DB8:2::/64 with lengths from /96 through /100.
- `2001:DB8:2::/64 eq 100` matches only prefixes of length /100 derived from 2001:DB8:2::/64.

Validation Rules

- Neither minimum length nor maximum length is valid without a mask length.
- For IPv4, the minimum length must be less than 32.
- For IPv6, the minimum length must be less than 128.
- The maximum length must be equal to or greater than the minimum length.

Invalid prefix set examples

Table 11: Invalid prefix set examples and validation errors

Invalid example	Validation rule error
`192.0.2.1 ge 16`	Minimum length (`ge`) specified without a mask length.
`192.0.2.2 le 16`	Maximum length (`le`) specified without a mask length.
`192.0.2.3/24 le 23`	Maximum length must be equal to or greater than the mask length; 23 is less than /24.
`192.0.2.0/24 ge 33`	For IPv4, the minimum length must be less than 32; 33 exceeds the valid range (0-32).
`192.0.2.0/25 ge 29 le 28`	Maximum length must be equal to or greater than the minimum length; 28 is less than 29.

ACL support in RPL prefix sets

Access Control List (ACL) type prefix set entries hold IPv4 or IPv6 prefix match specifications. Each specification includes an address and a wildcard mask. The address and wildcard mask use standard dotted-decimal IPv4 or colon-separated hexadecimal IPv6 notation.

The wildcard mask, also called an inverted mask, indicates which bits must match:

- binary 0 indicates a mandatory match, that is, bits set to 0 must match exactly, and
- binary 1 indicates a do not match condition, that is, bits set to 1 are ignored.

This format allows the prefix set to specify both contiguous and non-contiguous sets of bits that must be matched in a route.

Route distinguisher sets

Describes rd-sets and the supported RD formats used to match BGP VPN route distinguisher values, including ASN, IPv4, wildcard, and masked forms.

A route distinguisher set (rd-set) is a set in RPL that

- contains route distinguisher elements for BGP VPNs, and
- allows specification in IPv4 or ASN formats, with support for wildcards and subnet masks.

A route distinguisher (RD) is a 64-bit value that is prepended to an IPv4 address to create a globally unique Border Gateway Protocol (BGP) VPN IPv4 address.

Supported route distinguisher formats

RD set example

RD values can be defined using these formats.

Table 12: Supported RD formats

Format	Description	Example
`a.b.c.d:m:*`	BGP VPN RD in IPv4 format with a wildcard character.	`10.0.0.2:255.255.0.0:*`
`a.b.c.d/m:n`	BGP VPN RD in IPv4 format with a subnet mask.	`10.0.0.2:255.255.0.0:666`
`a.b.c.d:**`	BGP VPN RD in IPv4 format with a wildcard character.	`10.0.0.2:255.255.0.0`
`a.b.c.d:n`	BGP VPN RD in IPv4 format.	`10.0.0.2:666`
`asn:*`	BGP VPN RD in ASN format with a wildcard character.	`10002:255.255.0.0`
`asn:n`	BGP VPN RD in ASN format.	`10002:666`

```
rd-set rdset1
  10.0.0.0/8:*,
  10.0.0.0/8:777,
  10.0.0.0:*,
  10.0.0.0:777,
  65000:*,
```

```
  65000:777
end-set
```

Policy definitions

Describes how route policies are defined, named, attached, and removed, and explains how policy definitions provide the framework for evaluating and modifying routes.

A policy definition is a route policy that

- defines a named sequence of policy statements,
- starts with the **route-policy** keyword and a policy name, and
- ends with the **end-policy** keyword.

Policy definition syntax

Remove a policy definition

Policy names and binding

Use this syntax to define a policy:

```
route-policy <name>
    <policy statements>
end-policy
```

This example shows a policy that drops any route it encounters:

```
route-policy drop-everything
    drop
end-policy
```

Policy statements in the body determine how routes are evaluated and modified.

To remove a policy definition, use the **no route-policy <name>** command.

A policy name is an identifier that you use to bind a policy definition to a protocol attach point.

Example bind point in BGP configuration:

```
router bgp 2
  neighbor 10.1.1.1
    remote-as 65535
    address-family ipv4 unicast
      route-policy param-example(10, prefix_set1)
      route-policy param-example(20, prefix_set2)
```

When you attach a policy at an attach point, the system applies the policy for routes at that location.

Policy statements

Explains how policy statements control route evaluation through remarks, dispositions, actions, conditional logic, and policy calls executed in sequence.

A policy statement is a route policy instruction that

- defines how the policy evaluates, modifies, accepts, or rejects routes
- runs in sequence as it appears in the policy, and

- can include remarks, dispositions, actions, conditional logic, and policy calls.

Policy statement types

A route policy can include these policy statement types:

- Remark
- Disposition (drop, pass, done)
- Action, such as, set operations
- If (conditional logic)
- Apply (run another policy from the current policy)

Remarks
Describes how remarks document policies with persistent comment lines that are ignored by the parser and must be placed outside statements and inline sets.

A remark is a comment line in a policy that

- starts with `#` at the beginning of each line
- documents the policy, and
- is ignored by the policy language parser.

RPL remark usage and persistence

Example 1

Example 2

You typically place remarks between complete statements or between elements of a set. You cannot place remarks in the middle of statements or within an inline set definition.

Unlike comments preceded by an exclamation mark in the CLI, RPL remarks persist through device reboots and when you save and reload configurations.

```
# This is a simple one-line remark.
```

```
# This
# is a remark
# comprising multiple
# lines.
```

Dispositions
Explains how disposition statements determine whether routes pass, drop, or stop evaluation, and clarifies the default acceptance behavior when routes are modified.

A disposition statement is a policy statement that decides whether the policy accepts or rejects a route.

If the policy modifies a route, the route is accepted by default. If the policy does not modify a route, the route is dropped by default.

drop

Example

The **drop** statement explicitly discards the route. When a drop is executed:

- Policy execution stops immediately.
- No further statements are evaluated.

```
route-policy DROP-EXAMPLE
  pass
  drop
  pass
end-policy
```

In this example, the first **pass** is overridden by the **drop**. The second **pass** is never executed. The route is dropped.

An explicit **drop** overrides any previously executed **pass** statement or attribute modification.

pass

Example 1

Example 2

Use the **pass** statement to prevent a route from being dropped when no modifications have occurred. This statement lets your route continue through policy execution.

When policy execution completes, a modified route or a route with a pass disposition passes the policy.

```
route-policy PASS-ALL
  pass
end-policy
```

This policy also passes all routes because it modifies a route attribute.

```
route-policy SET-LPREF
  set local-preference 200
end-policy
```

done

Use the **done** statement to stop the policy execution and accept the route. When the system encounters **done** statement:

- The route passes through the policy.
- The policy does not evaluate any remaining statements.
- The policy preserves any modifications made before **done** statement.

Actions

Describes how actions modify route attributes through operations such as set and delete and how multiple actions can be grouped within a policy.

An action is a sequence of operations that modifies route attributes. Many actions use the **set** keyword.

In a route policy, actions can be grouped together. In this example, the route policy named *actions-example* comprises three actions: two set actions, and one delete action.

```
route-policy actions-example
  set med 217
  set community (12:34) additive
  delete community in (12:56)
end-policy
```

If statements

Explains how if, else, elseif, and nested conditions control policy behavior by evaluating Boolean expressions and executing actions or dispositions when conditions match.

An **if** statement is a conditional control structure in a route policy that

- evaluates a Boolean condition
- runs actions or dispositions when the condition is true, and
- can include an optional **else** clause to run alternative actions when the condition is false.

Basic example

Example with multiple actions

```
if as-path in as-path-set-1 then
  drop
endif
```

```
if origin is igp then
  set med 42
  prepend as-path 73.5 5
endif
```

> **Note**
>
> The CLI also supports **exit** as an alternative to **endif**.

Nested if behavior explanation

Table 13: Conditional variations in RPL

Construct	Purpose	Syntax Pattern	Example	Behavior
if	Executes statements when the condition is true.	`if <condition> then ... endif`	`if as-path in as-path-set-1 then drop endif`	The system runs the statements only when the condition evaluates to true.
if-else	Executes alternate statements when the condition is false.	`if <condition> then ... else ... endif`	`if med eq 8 then set community (12:34) additive else set community (12:56) additive endif`	The *then* block runs if the condition is true; otherwise, the *else* block runs.
if-elseif-else	Evaluates multiple conditions sequentially.	`if ... elseif ... else ... endif`	`if med eq 150 then set local-preference 10 elseif med eq 200 then set local-preference 60 elseif med eq 250 then set local-preference 110 else set local-preference 0 endif`	The system tests conditions in order. It executes the first condition that is true. If no condition matches, the *else* block runs.
Nested if	Allows conditional logic inside another conditional block.	`if ... then if ... then ... endif ... endif`		The system evaluates the inner condition only if the outer condition is true.

Construct	Purpose	Syntax Pattern	Example	Behavior
			`if community matches-any (12:34,56:78) then   if med eq 150 then   drop   endif   set local-preference 100   endif`	

In the nested example:

```
if community matches-any (12:34,56:78) then
  if med eq 150 then
    drop
  endif
  set local-preference 100
endif
```

- Routes matching community *12:34* or *56:78* receive local preference *100*.
- Routes that also have *MED 150* are dropped.
- The inner **if** statement executes only when the outer condition is satisfied.

Boolean conditions
Describes the Boolean operators, precedence rules, and grouping methods used to build true-or-false expressions for conditional policy evaluation.

A Boolean condition is an expression that evaluates to *true* or *false* and is used in an **if** statement.

Supported boolean operators

- `not` (negation)
- `and` (conjunction)
- `or` (disjunction)

Operator precedence

1. `not`
2. `and`
3. `or`

Table 14: Boolean condition examples in RPL

Condition type	Expression	Description
Simple condition	`med eq 42`	True only if the MED value is 42.

Condition type	Expression	Description
Negation	`not next-hop in (198.51.100.2)`	Evaluates to true if the next-hop is not 198.51.100.2.
Compound condition (and)	`med eq 42 and next-hop in (198.51.100.2)`	True only if both conditions are true.
Compound condition (or)	`origin is igp or origin is incomplete`	True if either condition is true.
Grouped condition	`(med eq 42 and next-hop in (198.51.100.2))`	Parentheses group conditions to control their evaluation.
Operator precedence example	`med eq 10 or not destination in (198.51.100.0/24) and community matches-any ([12..34]:[56..78])`	Evaluated using operator precedence: `not`, then `and`, then `or`.
Parentheses override precedence	`med eq 10 or ((not destination in (198.51.100.0/24)) and community matches-any ([12..34]:[56..78]))`	Parentheses show the order in which conditions are evaluated.
Grouped disjunction and conjunction example	`(origin is igp or origin is incomplete or not med eq 42) and next-hop in (198.51.100.2)`	Combines grouped disjunction with conjunction.

Apply statements

Explains how apply statements run another policy inline, preserve drop and pass behavior, and return control to the calling policy when evaluation continues.

An apply statement is a route policy statement that

- runs another policy from within the current policy
- can run a parameterized or unparameterized policy, and
- returns execution to the calling policy unless the applied policy drops the route.

How the apply statement works

Example: inline expansion of an applied policy

When a route policy uses an apply statement, the applied policy is evaluated inline, exactly as if its statements were written at that location in the calling policy.

Key points:

- The applied policy executes first, at the point where **apply** appears.
- Drop and pass semantics are preserved:
  - If the applied policy drops the route, execution stops immediately.

    - If the applied policy does not drop the route, control returns to the calling policy and evaluation continues.

- The final outcome is identical to manually expanding the applied policy in place.

The combination of *main-policy* and *filter-destination* policies functions the same as the expanded policy called *main-policy-expanded*.

Calling policy:

```
route-policy main-policy
  apply filter-destination
  if as-path neighbor-is '123' then
    pass
  endif
end-policy
```

Applied policy:

```
route-policy filter-destination
  if destination in (10.0.0.0/16 le 32) then
    drop
  endif
end-policy
```

Equivalent expanded policy:

```
route-policy main-policy-expanded
  if destination in (10.0.0.0/16 le 32) then
    drop
  endif
  if as-path neighbor-is '123' then
    pass
  endif
end-policy
```

Using the **apply** statement provides a structural convenience. It promotes reuse and readability without changing policy behavior. The router evaluates the logic as though the applied policy is written inline at the point of invocation.

Hierarchical policies using apply
Describes how hierarchical policies reuse common logic through apply statements and shows that referenced policy content behaves as though inserted inline.

A hierarchical policy is a route policy that

- references another policy by using the `apply` statement
- reuses common policy logic across multiple policies, and
- behaves as if the referenced policy content is inserted where you call `apply`.

Example policy that applies another policy:

```
route-policy check-as-1234
  if as-path passes-through '1234.5' then
    apply drop-everything
  else
    pass
  endif
end-policy
```

If the route passed through autonomous system 1234.5 before the system receives it, the policy applies the `drop-everything` policy and drops the route. If the route did not pass through autonomous system 1234.5 before the system receives it, the policy accepts the route without modification.

Equivalent policy with the applied content inline:

```
route-policy check-as-1234-prime
  if as-path passes-through '1234.5' then
    drop
  else
    pass
  endif
end-policy
```

Note

Avoid creating too many hierarchy levels to keep your policies easy to manage and understand.

Parameterization

Explains how parameterized policies accept variable inputs, support reuse through apply and attach points, and use local and global parameters during policy evaluation.

Parameterization is a route policy capability that

- lets you define a policy that accepts one or more parameters
- uses parameters that start with `$` and contain only alphanumeric characters, and
- allows you to substitute parameters into any attribute that accepts a parameter.

Policy parameterization

In addition to reusing policies through the **apply** statement, you can define policies to accept parameters. Substitute these parameters into attributes within the policy as needed.

Example 1:

```
route-policy param-example ($mytag)
  set community (1234:$mytag) additive
end-policy
```

In this example, the policy *param-example* takes one parameter, *$mytag*, which is used as part of a 16-bit community tag.

Example 2:

In this example, the policy takes two parameters, *$mymed* and *$prefixset*.

```
route-policy param-example ($mymed, $prefixset)
  if destination in $prefixset then
    set med $mymed
  endif
end-policy
```

In this case, a MED value and a prefix-set name are passed as parameters and substituted into the policy statements.

Reuse using apply with parameters

You can reuse a parameterized policy with different values by supplying arguments in the **apply** statement.

Example:

```
route-policy origin-10
  if as-path originates-from '10.5' then
    apply param-example(10.5)
  else
    pass
  endif
end-policy
route-policy origin-20
  if as-path originates-from '20.5' then
    apply param-example(20.5)
  else
    pass
  endif
end-policy
```

When you provide values in the **apply** statement, the policy *param-example* uses those values. If you change the definition of *param-example*, *origin-10* and *origin-20* automatically change their behavior.

The *origin-10* policy adds the community *1234:10* to routes that originate from autonomous system 10. Similarly, *origin-20* adds the community *1234:20* to routes originating from autonomous system 20.

Parameterization at attach points

The system also supports parameterization at attach points. You can supply parameters at attach points when you bind the policy to a protocol configuration.

Example:

```
router bgp 2
  neighbor 192.0.2.1
    remote-as 3
    address-family ipv4 unicast
      route-policy param-example(10, prefix_set1)
      route-policy param-example(20, prefix_set2)
```

In this case, the parameterized policy *param-example* is expanded using the values provided in the route-policy configuration at the attach point.

Global parameterization

RPL supports systemwide global parameters that you can reference in policy definitions.

To define global parameters, use this format:

```
policy-global
  glbpathtype 'ebgp'
  glbtag '100'
end-global
```

Use these values in a policy definition the same way you use local parameters.

Example:

```
route-policy globalparam
  if path-type is $glbpathtype then
    set tag $glbtag
```

```
  endif
end-policy
```

Local and global parameter precedence

- If you define a parameter in your policy with the same name as a global parameter, the local parameter takes precedence and hides the global parameter.
- You cannot delete a global parameter that any policy references.

Policy execution and semantics

Describes how RPL evaluates statements in order, applies inline policy behavior, enforces default drop semantics, delays attribute changes until testing completes, and handles repeated attribute modifications.

Policy execution is the process of evaluating a route policy that

- processes policy statements sequentially in the order they appear in the configuration
- runs an applied policy and then returns to the calling policy unless the route is dropped, and
- treats an applied policy as if its contents are inserted inline at the point where the policy uses **apply** statement.

Order of policy execution

The system processes the policy statements sequentially, as they appear in the configuration.

If a policy uses the **apply** statement, execution proceeds into the applied policy. It then returns to the calling policy, unless the route is dropped.

When you apply a policy, its contents are treated as if they are inserted inline at the point where you use the **apply** statement.

Example:

```
route-policy one
  set weight 100
end-policy

route-policy two
  set med 200
end-policy

route-policy three
  apply two
  set community (2:666) additive
end-policy

route-policy four
  apply one
  apply three
  pass
end-policy
```

Policy *four* is equivalent to:

```
route-policy four-equivalent
  set weight 100
  set med 200
```

```
  set community (2:666) additive
  pass
end-policy
```

Note

You may remove the **pass** statement while preserving equivalent behavior, depending on the context.

Default drop disposition

By default, if you do not add a policy action, the route is dropped. All route policies have an implicit drop disposition, unless the route:

- is modified by a policy action, or
- is explicitly passed using the **pass** statement.

Consider this example:

```
route-policy two
  if destination in (10.0.0.0/8 ge 8 le 32) then
    set local-preference 200
  endif
end-policy

route-policy one
  apply two
end-policy
```

Although policy *one* does not explicitly pass routes, it inherits the disposition from policy *two*. Routes in network 10 that are modified by setting the local preference are passed; all other routes are dropped.

Attribute modification timing

The system does not modify route attributes until all tests in your policy are complete. All comparison operators evaluate the original route data. If you make intermediate modifications, they do not affect subsequent conditional tests.

Example:

```
if med eq 12 then
  set med 42
  if med eq 42 then
    drop
  endif
endif
```

The second test evaluates the original MED value, not the modified value. Therefore, the **drop** statement is never executed.

Multiple modifications of the same attribute

If you modify an attribute multiple times within a policy, the result depends on the attribute type.

Scalar attributes

For attributes that contain only one value, such as MED, your last assignment takes effect.

Example:

```
set med 9
set med 10
set med 11
set med 12
```

The resulting MED value is 12.

Example with conditional modification:

```
set med 8
if community matches-any cs1 then
  set local-preference 122
  if community matches-any cs2 then
    set med 12
  endif
endif
```

The route has a MED of 8 unless your policy matches both *cs1* and *cs2*. In this case, MED is 12.

Cumulative attributes

Additive and array attributes support cumulative behavior rather than replacing prior values.

Additive attribute example:

```
route-policy community-add
  set community (10:23)
  set community (10:24) additive
  set community (10:25) additive
end-policy
```

Your resulting community list contains all three values: *10:23*, *10:24*, and *10:25*.

Array-type attribute example:

```
route-policy prepend-example
  prepend as-path 2.5 3
  prepend as-path 666.5 2
end-policy
```

This configuration prepends *666.5 666.5 2.5 2.5 2.5* to the AS path. Each prepend operation is applied, because the AS path is an array-type attribute.

Policy verification and integrity

Explains how policy verification checks definitions, parameters, references, recursion, and attached objects to ensure policies are valid, consistent, and safe to activate.

Policy verification and integrity is a set of checks that

- validates that policies are well formed, internally consistent, and safe to attach to protocol configuration
- enforces value and parameter validation during policy definition and policy attachment, and
- prevents invalid configurations such as recursive policy references and removal of policies or sets that are in use at an attach point.

Range checking

As policies are defined, the system performs simple verifications such as range checking of values. For example, when setting the Multi-Exit Discriminator (MED) attribute, the system verifies whether the value is within the proper range for the attribute.

Range checking cannot validate parameter specifications at definition time if parameter values are not defined. These are verified later when the policy is attached at an attach point.

At attach time, all policies must be well formed. All referenced sets and policies must be defined and contain valid values. Each parameter value must be within its proper range.

Validation at policy attach time

When you attach a policy, the system validates parameter specifications. Review these validation points:

- Define all referenced policies and sets.
- Make sure parameter values are valid.
- Confirm that all policies are well-defined.
- Verify recursive definitions.
- Make sure the number of parameters is correct.

Parameter validation ensures that a policy is complete and valid before it becomes active at an attach point.

Recursive definition checks

The policy repository checks for recursive definitions. This prevents policies from directly or indirectly being applied in a way that causes invalid configuration.

The system enforces recursive definition checks before it allows policies to be attached.

Incomplete policy and set references

If you have not attached a policy, you can refer to sets or policies that do not exist yet. This gives you flexibility to define policies that refer to sets or policies you plan to create later.

For example, a policy may reference another policy using the **apply** statement even if the referenced policy does not yet exist. Similarly, a policy statement may refer to a set that has not been defined.

However, when you attach a policy, the system enforces the existence of all referenced policies and sets. If a policy references an undefined set or policy at attach time, the system rejects the configuration attempt.

Example:

If policy *sample* references policy *bar*, and *bar* does not exist, attempting to attach *sample* at an inbound BGP policy results in an error.

Null policy behavior

A null policy is a policy block that exists but contains no statements, actions, or dispositions.

Example:

```
route-policy bar
end-policy
```

This is a valid policy block. However, because it neither modifies the route nor includes a **pass** statement, the default drop disposition applies. As a result, the system drops all routes evaluated by this policy.

Protection for referenced policies and sets

You cannot remove a route policy or set that is currently in use at an attach point. If you try to remove a policy or set that is in use, the system displays an error message.

This behavior helps you avoid undefined references and keeps your configuration consistent.

Aggregation policies

Describes how aggregation policies control aggregate route generation, filter or suppress contributing routes, modify aggregate attributes, and apply deterministic processing to more-specific routes.

An aggregation policy is a route policy that

- you attach at the aggregation attach point
- controls whether the system generates an aggregate route based on the presence of more-specific routes, and
- can modify valid BGP attributes on the aggregate route while controlling how more-specific routes contribute.

Aggregation attach point behavior

The aggregation attach point generates an aggregate route for advertisement if certain subcomponents of the aggregate are present.

Policies attached at this attach point can:

- determine whether the system generates the aggregate, and
- set valid BGP attributes on the aggregated route.

The system generates an aggregate if any routes evaluated by the named policy pass the policy.

Example:

```
route-policy sample
  if destination in (10.0.0.0/8 ge 8 le 25) then
    set community (10:33)
  endif
  if destination in (10.2.0.0/24) then
    drop
  endif
  if destination in (10.1.0.0/24) then
    suppress-route
  endif
end-policy

router bgp 2
address-family ipv4
  aggregate-address 10.0.0.0/8 route-policy sample
```

In this example:

- The system generates the aggregate 10.0.0.0/8 if any component routes in the range 10.0.0.0/8 ge 8 le 25 pass the policy.
- The system drops routes matching 10.2.0.0/24. These routes do not contribute.
- The system suppresses routes matching 10.1.0.0/24.
- The system may set the community 10:33 on the aggregate.

Attribute modification on aggregates

Policies attached at the aggregation attach point may set valid BGP attributes on the aggregate route. Actions such as setting community values or MED affect the aggregate the system generates.

In the policy language, you can set attributes on the aggregate using normal action operations. The system generates the aggregate if at least one evaluated route passes the policy.

Suppress-route behavior

When you do not set **summary-only**, the system advertises both the aggregate and more-specific routes.

You can selectively filter specific component routes using the **suppress-route** keyword.

In the policy language:

- Select the route and set the suppress flag to control suppression.
- This approach replaces the traditional suppress map mechanism.

Cumulative effects on aggregate attributes

Each time the aggregation policy matches a more-specific route, it may modify aggregate attributes through set operations. These effects accumulate with each match.

Example:

```
route-policy
        bumping-aggregation
        set med +5
        end-policy
```

Each contributing more-specific route increments the aggregate MED by 5.

- If 3 matching more-specific routes exist, the aggregate MED becomes *default plus 15*.
- If 17 matching more-specific routes exist, the aggregate MED becomes *default plus 85*.

Drop behavior in aggregation

A **drop** in an aggregation policy does not prevent the system from generating the aggregate. It only prevents the current more-specific route from contributing to the aggregate.

The system generates the aggregate when any other more-specific route passes the policy. A single passing more-specific route is sufficient for the system to create the aggregate.

Deterministic but unspecified processing order

The system applies the aggregation policy to prefix paths in a deterministic, but unspecified, order. The system processes routes in the same order each time for a given set of routes, but the order cannot be predicted.

5 Apply and Manage Routing Policies

Topics:

Explains how to attach, validate, modify, and edit routing policies across supported protocol contexts, including BGP, OSPF, OSPFv3, and IS-IS attach points.

You can define routing policies independently and attach them to specific protocol contexts, such as routing processes, address families, or neighbors. Use the policy name as a reference to reuse the policy at multiple attach points without redefining it. After you attach a policy, the system evaluates routes processed at that location. Provide arguments to parameterized policies at attach time for flexible behavior.

When you attach a policy, the system verifies that the referenced objects exist, parameters are correct, and the policy is well formed. This ensures you can deploy routing policies using a modular and scalable framework.

Attach a routing policy

Explains how policy attachment activates a named policy at a supported attach point, validates references and protocol compatibility, and supports parameterized use in protocol-specific contexts.

Attaching a routing policy is the configuration action that

- references a policy name at a supported protocol attach point
- activates the policy so the system applies it to routes processed at that attach point, and
- validates that all referenced sets and policies exist, rejecting the attachment if dependencies are undefined.

A policy attachment can also include arguments to parameterize the policy at the attach point.

Association with protocol context

Policies become effective only after they are attached to a routing protocol entity, such as a neighbor, address family, or redistribution point. The attach point defines the association between the protocol context and the named policy, enabling the protocol to apply the policy to relevant routes, for example during route import or export processing.

Protocol attributes and attachment validation

Each routing protocol defines a specific set of route attributes that can be matched or modified by policy. During attachment, the system verifies that the policy operates only on attributes that are valid for the selected protocol context. The protocol configuration rejects attempts to attach policies that perform unsupported operations.

For example, a policy that matches BGP community attributes cannot be attached to OSPF, and a policy that sets IS-IS-specific attributes cannot be applied to BGP routes. Similarly, when a policy that is already in use is modified, the system verifies that the changes remain compatible with all existing attachment points.

Attach points operating on RIB routes

Some attach points operate on routes in the Routing Information Base (RIB) rather than native protocol routes. In these cases, the policy may match attributes from the RIB representation and set attributes relevant to the target protocol, depending on the direction of processing.

Redistribution and mixed attribute operations

Some policy attach points are used during route redistribution, where routes move between protocols through the common RIB representation. These scenarios can involve matching attributes from the source representation and setting attributes for the destination protocol. As a result, policies applied at redistribution points may combine operations across attribute domains, depending on the processing stage and protocol context.

BGP policy attach points

Describes the BGP locations where policies can control route selection, advertisement, injection, filtering, retention, labeling, operational views, and protocol-specific processing behavior.

A BGP policy attach point is a specific location in BGP configuration where you can attach a routing policy that

- applies the policy to routes processed at that BGP processing stage, and
- controls how BGP evaluates or modifies routes by using BGP attributes and operators relevant to that attach point.

BGP policy attach points by function

Use these BGP policy attach points to control how BGP selects, advertises, imports, and manages routes:

- Path selection and advertisement

  - additional-path
  - default originate
  - neighbor export
  - neighbor import
- Route injection
  - network
  - redistribute
- Display, accounting, and operations
  - show bgp
  - table policy
  - clear-policy
  - debug
- VPN route movement and retention
  - import
  - export
  - retain route-target
- Label and ORF controls
  - allocate-label
  - label-mode
  - neighbor-orf
- Next-hop event control: next-hop
- Stability control: dampening

additional-path

The additional-path attach point controls whether BGP selects additional paths so a BGP speaker can send multiple paths for the same prefix.

Example:

```
router bgp 100
  address-family ipv4 unicast
    additional-paths selection route-policy add-path-policy
```

This table summarizes the BGP attributes and operators available for the additional-path attach point.

Table 15: Attributes and operators for additional-path

Attach Point	Attribute	Match	Set
aggregation	as-path	in	–
		is-local	
		length	
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–
	community	is-empty	set
		matches-any	set additive
		matches-every	delete in
			delete not in
			delete all
	destination	in	–
	extcommunity cost	–	set
			set additive
	local-preference	is, ge, le, eq	set
	med	is, eg, ge, le	setset +set -
	next-hop	in	set
	origin	is	set
	source	in	–
	suppress-route	–	suppress-route
	weight	–	set

dampening

The dampening attach point controls BGP route-dampening behavior by using a policy to set dampening attributes for routes.

Policy behavior:

- The last **set dampening** statement encountered takes effect.
- If the policy executes **drop**, the route does not contribute to dampening.
- If the policy executes **pass** or **done** without **set dampening**, the default dampening parameters apply.
- If an applied policy executes **pass**, execution returns to the calling policy.

- If an applied policy executes **done**, policy execution stops and accepts the route.

Example:

```
route-policy sample_damp
  if destination in (0.0.0.0/0 ge 25) then
    set dampening halflife 30 others default
  else
    set dampening halflife 20 others default
  endif
end-policy
router bgp 2
  address-family ipv4 unicast
    bgp dampening route-policy sample_damp
...
```

This table summarizes the BGP attributes and operators available for the dampening attach point.

Table 16: Attributes and operators for dampening

Attach Point	Attribute	Match	Set
dampening	as-path	in	–
		is-local	
		length	
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–
	community	is-empty	–
		matches-any	
		matches-every	
	dampening	–	set dampening
	destination	in	–
	local-preference	is, ge, le, eq	–
	med	is, eg, ge, le	–
	next-hop	in	–
	origin	is	–
	source	in	–

default originate

The default originate attach point conditionally generates and advertises the default route 0.0.0.0/0 to a peer based on whether routes in the RIB pass the attached policy.

Example:

```
route-policy sample-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 32) then
    pass
  endif
end-policy

router bgp 2
  neighbor 10.0.0.1
    remote-as 3
    address-family ipv4 unicast
      default-originate policy sample-originate
...
```

This table summarizes the BGP attributes and operators available for the default originate attach point.

Table 17: Attributes and operators for default originate

Attach Point	Attribute	Match	Set
default originate	med	–	set
			set +
			set -
	rib-has-route	in	–

neighbor export

The neighbor export attach point selects which BGP routes BGP sends to a peer or peer group by running candidate routes through the attached policy. Routes that pass are sent, and the policy can modify route attributes before BGP sends the update.

Example:

```
route-policy sample-export
  if community matches-any (2:[100-200]) then
    set med 100
    set community (2:666)
  else
    set med 200
    set community (2:200)
  endif
end-policy

router bgp 2
  neighbor 10.0.0.5
    remote-as 3
  address-family ipv4 unicast
    route-policy sample-export out
...
```

This table summarizes the BGP attributes and operators available for the neighbor export attach point.

Table 18: Attributes and operators for neighbor export

Attach Point	Attribute	Match	Set
neighbor-out	as-path	in	prepend
		is-local	prepend most-recent
		length	remove as-path private-as
		–	replace
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–
	communitycommunity with ‘peeras’	is-empty	set
		matches-any	set additive
		matches-every	delete-in
			delete-not-in
			delete-all
	destination	in	–
	extcommunity cost	–	set
			set additive
	extcommunity rt	is-empty	set
		matches-any	additive
		matches-every	delete-in
		matches-within	delete-not-in
			delete-all
	extcommunity soo	is-empty	–
		matches-any	
		matches-every	
		matches-within	
	local-preference	is, ge, le, eq	set
	med	is, eg, ge, le	set
			set +
			set -
			set max-unreachable
			set igp-cost

Attach Point	Attribute	Match	Set
	next-hop	in	set set self
	origin	is	set
	path-type	is	–
	rd	in	–
	route-aggregated	route-aggregated	–
	source	in	–
	unsuppress-route	–	unsuppress-route
	vpn-distinguisher	–	set

neighbor import

The neighbor import attach point controls which routes BGP accepts from a peer by running received routes through the attached policy. Routes that pass become candidates in the BGP RIB.

When you change an import policy, BGP must re-evaluate routes from that peer against the new policy. The **bgp auto-policy-soft-reset** option can automate this when soft reconfiguration is enabled or route refresh is negotiated.

Example:

```
route-policy sample_import
  if community matches-any (3:100) then
    set local-preference 100
    set community (2:666)
  else
    set local-preference 200
    set community (2:200)
  endif
end-policy

router bgp 2
  neighbor 10.0.0.1
    remote-as 3
    address-family ipv4 unicast
      route-policy sample_import in
```

This table summarizes the BGP attributes and operators available for the neighbor import attach point.

Table 19: Attributes and operators for neighbor import

Attach Point	Attribute	Match	Set
neighbor-in	as-path	in	prepend
		is-local	prepend most-recent
		length	remove as-path private-as
		NA	replace
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–
	communitycommunity with ‘peeras’	is-empty	set
		matches-any	set additive
		matches-every	delete-in
			delete-not-in
			delete-all
	destination	in	–
	extcommunity cost	–	set
			set additive
	extcommunity rt	is-empty	set
		matches-any	additive
		matches-every	delete-in
		matches-within	delete-not-in
			delete-all
	extcommunity soo	is-empty	–
		matches-any	
		matches-every	
		matches-within	
	local-preference	is, ge, le, eq	set
	med	is, eg, ge, le	set
			set +
			set -
	next-hop	in	set
			set peer address

Attach Point	Attribute	Match	Set
	origin	is	set
	route-aggregated	route-aggregated	NA
	source	in	–
	weight	–	set

network

The network attach point controls how BGP injects routes from the RIB into BGP. A policy at this attach point can set valid BGP attributes on injected routes.

Example:

```
route-policy NetworkControl
  if destination in (0.0.0.0/0 ge 25) then
    set community (no-export) additive
  endif
end-policy

router bgp 2
  address-family ipv4 unicast
    network 172.16.0.5/27 route-policy NetworkControl
```

This table summarizes the BGP attributes and operators available for the network attach point.

Table 20: Attributes and operators for network

Attach Point	Attribute	Match	Set
network	as-path	–	prepend
	community	–	set
			set additive
			delete-in
			delete-not-in
			delete-all
	destination	in	–
	extcommunity cost	–	set
			set additive
	mpls-label	route-has-label	–
	local-preference	–	set
	med	–	set
			set+
			set-
	next-hop	in	set
	origin	–	set
	route-type	is	–
	tag	is, ge, le, eq	–
	weight	–	set

redistribute

The redistribute attach point injects routes from other routing protocols into OSPF by selecting routes and setting OSPF parameters such as cost and metric type.

Example:

```
route-policy OSPF-redist
  set metric-type type-2
  if tag eq 10 then
    set ospf cost 100
  elseif tag eq 20 then
    set ospf cost 200
  else
    drop
  endif
end-policy


router ospf 1
  redistribute isis instance_10 policy OSPF-redist
```

This table summarizes the BGP attributes and operators available for the redistribute attach point.

Table 21: Attributes and operators for redistribute

Attach Point	Attribute	Match	Set
redistribute	as-path	–	prepend
	community	–	set
			set additive
			delete in
			delete not in
			delete all
	destination	in	–
	extcommunity cost	–	setset additive
	local-preference	–	set
	med	–	set
			set+
			set-
	next-hop	in	set
	origin	–	set
	mpls-label	route-has-label	–
	route-type	is	–
	tag	is, eq, ge, le	–
	weight	–	set

show bgp

The show bgp attach point displays BGP routes that are not dropped by the attached policy.

Example:

```
route-policy sample-display
  if med eq 5 then
    pass
  endif
end-policy

show bgp route-policy sample-display
```

This table summarizes the BGP attributes and operators available for the show bgp attach point.

Table 22: Attributes and operators for show bgp

Attach Point	Attribute	Match	Set
show bgp	as-path	in	–
		is-local	
		length	
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–
	community	is-empty	–
		matches-any	
		matches-every	
	destination	in	–
	extcommunity rt	is-empty	–
		matches-any	
		matches-every	
		matches-within	
	extcommunity soo	is-empty	–
		matches-any	
		matches-every	
		matches-within	
	med	is, eg, ge, le	–
	next-hop	in	–
	origin	is	–
	source	in	–

table policy

The table policy attach point sets traffic-index values on routes as the system installs routes into the global routing table. This supports BGP policy accounting by classifying routes and tracking traffic counters per class.

Example:

```
route-policy sample-table
  if as-path originates-from '10.33' then
    set traffic-index 10
  elseif as-path originates-from '11.60' then
    set traffic-index 11
```

```
  endif
end-policy

router bgp 2
  address-family ipv4 unicast
    table-policy sample-table
```

import

The import attach point controls import of routes from the global VPN IPv4 table into a VRF.

Example:

```
route-policy bgpvrf_import
  if extcommunity rt matches-any (10:91) then
    set next-hop 172.16.0.1
  elseif extcommunity rt matches-every (11:92) then
    set next-hop 172.16.0.2
  elseif extcommunity soo matches-any (10:111111, 10:111222) then
    pass
  endif
end-policy

vrf vrf_import
  address-family ipv4 unicast
    import route-policy bgpvrf_import
```

export

The export attach point controls export of routes from a VRF to the global VPN IPv4 table, including conditional route-target handling.

Example:

```
route-policy bgpvrf_export
  if destination in (172.16.1.0/24) then
    set extcommunity rt (10:101)
    set weight 211
  elseif origin is egp then
    set local-preference 212
    set extcommunity rt (10:101)
  endif
  set extcommunity rt (10:111222) additive
end-policy

vrf vrf-export
  address-family ipv4 unicast
    export route-policy bgpvrf-export
```

retain route-target

The retain route-target attach point retains VPN routes based only on route target extended communities. This can reduce the route scale at a route reflector or support IPv4 VPN route retention at an autonomous system boundary router (ASBR).

Example:

```
extcommunity-set rt rtset1
  0:615,
  10:6150,
  192.0.2.15:15
```

```
end-set


route-policy retainer
  if extcommunity rt matches-any rtset1 then
    pass
  endif
end-policy


router bgp 2
  address-family vpnv4 unicast
    retain route-target route-policy retainer
```

This table summarizes the BGP attributes and operators available for the retain route-target attach point.

Table 23: Attributes and operators for retain route-target

Attach Point	Attribute	Match	Set
retain-rt	extcommunity rt	is-empty	–
		matches-any	
		matches-every	
		matches-within	

allocate-label

The allocate-label attach point decides whether BGP allocates a label when it sends updates for the IPv4 labeled-unicast address family. This attach point typically uses AS path match criteria and supports pass and drop.

This table summarizes the BGP attributes and operators available for the allocate-label attach point.

Table 24: Attributes and operators for allocate-label

Attribute	Match	Set
as-path	in	–
	is-local	
	length	
	neighbor-is	
	originates-from	
	passes-through	
	unique-length	
as-path-length	is, ge, le, eq	–
as-path-unique-length	is, ge, le, eq	–
community	is-empty	–
	matches-any	
	matches-every	
destination	in	–

Attribute	Match	Set
label	–	set
local-preference	is, ge, le, eq	–
med	is, eg, ge, le	–
next-hop	in	–
origin	is	–
source	in	–

label-mode

The label-mode attach point selects a label mode based on match criteria such as prefix or community. This attach point is commonly used to choose per-CE, per-VRF, or per-prefix label modes and supports pass and drop.

neighbor-orf

The neighbor-orf attach point filters inbound BGP route updates by using prefix-based matching and sends the same prefix filters to upstream neighbors as outbound route filter (ORF) entries.

Example:

```
prefix-set orf-preset
  172.16.1.0/24,
  172.16.5.0/24,
  172.16.11.0/24
end-set

route-policy policy-orf
  if orf prefix in orf-preset then
    drop
  endif
  if orf prefix in (172.16.3.0/24, 172.16.7.0/24, 172.16.13.0/24) then
    pass
  endif

router bgp 2
  neighbor 1.1.1.1
    remote-as 3
    address-family ipv4 unicast
      orf route-policy policy-orf
```

This table summarizes the BGP attributes and operators available for the neighbor-orf attach point.

Table 25: Attributes and operators for neighbor-orf

Attach Point	Attribute	Match	Set
neighbor-orf	orf-prefix	in	Not applicable

next-hop

The next-hop attach point controls which RIB next-hop up or down notifications BGP processes by matching prefixes and protocols. This attach point is commonly used with next-hop tracking to monitor non-BGP routes.

Example:

```
route-policy nxthp_policy_A
  if destination in (10.0.0.0/8) and protocol in (static, connected) then
    pass
  endif
end-policy


router bgp 2
  address-family ipv4 unicast
    nexthop route-policy nxthp_policy_A
```

This table summarizes the BGP attributes and operators available for the next-hop attach point.

Table 26: Attributes and operators for next-hop

Attach Point	Attribute	Match	Set
next-hop	destination	in	–
	protocol	is,in	–
	source	in	–

clear-policy

The clear-policy attach point controls whether clear bgp operations clear flap statistics based on AS path match criteria.

Example:

```
as-path-set my-as-set
  ios-regex '_12$',
  ios-regex '_13$'
end-set

route-policy policy_a
  if as-path in my-as-set then
    pass
  else
    drop
  endif
end-policy

clear bgp ipv4 unicast flap-statistics route-policy policy_a
```

This table summarizes the BGP attributes and operators available for the clear-policy attach point.

Table 27: Attributes and operators for clear-policy

Attach Point	Attribute	Match	Set
clear-policy	as-path	in	–
		is-local	
		length	
		neighbor-is	
		originates-from	
		passes-through	
		unique-length	
	as-path-length	is, ge, le, eq	–
	as-path-unique-length	is, ge, le, eq	–

debug

The debug attach point filters BGP debug output by matching route prefixes.

Example:

```
route-policy policy_b
  if destination in (10.0.0.0/8) then
    pass
  else
    drop
  endif
end-policy

debug bgp update policy_b
```

This table summarizes the BGP attributes and operators available for the debug attach point.

Table 28: Attributes and operators for debug

Attach Point	Attribute	Match	Set
debug	destination	in	–

Restricted BGP operations by attach point

Provides restrictions for BGP operations that are unavailable or conditionally allowed at specific attach points, including cases limited to eBGP contexts.

Some BGP route attributes and operations are not available at every BGP attach point. For example, **set med igp-cost** requires a configured IGP cost.

Table 29: Restricted BGP operations by attach point

Command	Import	Export	Aggregation	Redistribution
prepend as-path most-recent	eBGP only	eBGP only	Not applicable	Not applicable
replace as-path	eBGP only	eBGP only	Not applicable	Not applicable

Command	Import	Export	Aggregation	Redistribution
set med igp-cost	Forbidden	eBGP only	Forbidden	Forbidden
set weight	Not applicable	Forbidden	Not applicable	Not applicable
suppress	Forbidden	Forbidden	Not applicable	Forbidden

OSPF policy attach points

Describes the OSPF locations where policies can filter summaries, inject default routes, control redistribution, and influence SPF-related processing.

An OSPF policy attach point is a specific location in an OSPF configuration where you can attach a routing policy that

- applies the policy to routes processed at that OSPF processing stage, and
- controls how OSPF evaluates or modifies routes by using OSPF attributes and operators relevant to that attach point.

OSPF supports routing policy attachment at multiple points in the protocol workflow, where policies can influence route origination, redistribution, inter-area processing, and shortest path first (SPF) behavior:

- area-in
- area-out
- default-information originate
- distribute-list in
- redistribute
- spf-prefix-priority

area-in

The area-in attach point filters inbound OSPF type-3 summary LSAs by using prefix-based matching.

In this example, the policy drops summary LSAs that match specific *10.105.x.x/24* prefixes and passes summary LSAs that match specific *10.106.x.x/24* prefixes.

Example:

```
route-policy OSPF-area-in
  if destination in (10.105.3.0/24, 10.105.7.0/24, 10.105.13.0/24) then
    drop
  endif
  if destination in (10.106.3.0/24, 10.106.7.0/24, 10.106.13.0/24) then
    pass
  endif
end-policy


router ospf 1
  area 1
    route-policy OSPF-area-in in
```

area-out

The area-out attach point filters outbound OSPF type-3 summary LSAs by using prefix-based matching.

In this example, the policy drops summary LSAs that match specific prefixes and passes summary LSAs that match specific prefixes so OSPF can announce them.

Example:

```
route-policy OSPF-area-out
  if destination in (10.105.3.0/24, 10.105.7.0/24, 10.105.13.0/24) then
    drop
  endif
  if destination in (10.105.3.0/24, 10.105.7.0/24, 10.105.13.0/24) then
    pass
  endif
end-policy


router ospf 1
  area 1
    route-policy OSPF-area-out out
```

default-information originate

The default-information originate attach point conditionally injects the default route 0.0.0.0/0 into the OSPF link-state database by evaluating the attached policy against routes in the local RIB. If any routes pass the policy, the system inserts the default route into the link-state database.

Example:

```
route-policy ospf-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 25) then
    pass
  endif
end-policy


router ospf 1
  default-information originate policy ospf-originate
```

distribute-list in

The distribute-list in attach point filters OSPF prefixes by using a route policy. You can configure **distribute-list in route-policy** at the OSPF instance, area, and interface levels.

Route policies at this attach point support:

- **destination**
- **rib-metric**

Route policies at this attach point do not support **set** commands.

Example 1:

```
route-policy DEST
  if destination in (10.10.10.10/32) then
    drop
  else
    pass
  endif
end-policy
```

Example 2:

```
route-policy METRIC
  if rib-metric ge 10 and rib-metric le 19 then
    drop
  else
    pass
  endif
end-policy
```

Example 3:

```
prefix-set R-PFX
  10.10.10.30
end-set
route-policy R-SET
  if destination in R-PFX and rib-metric le 20 then
    pass
  else
    drop
  endif
end-policy
```

redistribute

The redistribute attach point injects routes from other routing protocols into the OSPF link-state database. The policy selects which routes to inject and sets OSPF attributes such as *metric type* and *cost* by using **set metric-type** and **set ospf cost**.

In this example, the policy sets metric type type-2 for all redistributed routes, sets the OSPF cost based on the IS-IS route tag, and drops routes that do not match tag 10 or 20.

Example:

```
route-policy OSPF-redist
  set metric-type type-2
  if tag eq 10 then
    set ospf cost 100
  elseif tag eq 20 then
    set ospf cost 200
  else
    drop
  endif
end-policy


router ospf 1
  redistribute isis instance_10 policy OSPF-redist
```

OSPF attributes and operators

Lists the attributes, match operators, and set operations supported at each OSPF policy attach point.

This table summarizes the OSPF attributes and operators for each attach point.

Table 30: OSPF attributes and operators

Attach point	Attribute	Match	Set
default-information originate	ospf-metric	–	set
	metric-type	–	set
	tag	–	set
	rib-has-route	in	–
redistribute	destination	in	–
	metric-type	–	set
	ospf-metric	–	set
	next-hop	in	–
	mpls-label	route-has-label	–
	rib-metric	is, le, ge, eq	n/a
	route-type	is	–
	tag	is, eq, ge, le	set
area-in	destination	in	–
area-out	destination	in	–
spf-prefix-priority	destination	in	n/a
	spf-priority	n/a	set
	tag	is, le, ge, eq	n/a

OSPFv3 policy attach points

Describes the OSPFv3 locations where policies can control default route origination and redistribution behavior by using supported OSPFv3 attributes and operators.

An OSPFv3 policy attach point is a specific location in an OSPFv3 configuration where you can attach a routing policy that

- applies the policy to routes processed at that OSPFv3 processing stage, and
- controls how OSPFv3 evaluates or modifies routes by using OSPFv3 attributes and operators relevant to that attach point.

OSPF supports routing policy attachment at multiple points in the protocol workflow, where policies can influence route origination and redistribution.

- default-information originate
- redistribute

default-information originate

The default-information originate attach point conditionally injects the default route 0.0.0.0/0 into the OSPFv3 link-state database by evaluating the attached policy against routes in the local RIB. If any routes pass the policy, the system inserts the default route into the link-state database.

redistribute

The redistribute attach point injects routes from other routing protocols into the OSPFv3 link-state database. The attached policy selects which routes to inject and sets OSPFv3 attributes such as *metric type* and *cost* by using **set metric-type** and **set ospf cost** statements.

In this example, the policy sets metric type *type-2* for all redistributed routes, sets the cost based on the route tag, and drops routes with tags other than 10 or 20.

Example:

```
route-policy OSPFv3-redist
  set metric-type type-2
  if tag eq 10 then
    set extcommunity cost 100
  elseif tag eq 20 then
    set extcommunity cost 200
  else
    drop
  endif
end-policy


router ospfv3 1
  redistribute bgp 15 policy OSPFv3-redist
```

OSPFv3 attributes and operators

Lists the attributes, match operators, and set operations supported at each OSPFv3 policy attach point.

This table summarizes the OSPF attributes and operators for each attach point.

Table 31: OSPFv3 attributes and operators

Attach Point	Attribute	Match	Set
default-information originate	ospf-metric	–	set
	metric-type	–	set
	tag	–	set
	rib-has-route	in	–
redistribute	destination	in	–
	ospf-metric	–	set
	metric-type	–	set
	route-type	is	–
	tag	is, eq, ge, le	–

IS-IS policy attach points

Describes the IS-IS locations where policies can control default route origination and inter-area prefix propagation between levels.

An IS-IS policy attach point is a specific location in an IS-IS configuration where you can attach a routing policy that

- applies the policy to routes processed at that IS-IS processing stage, and
- controls how IS-IS evaluates or modifies routes by using IS-IS attributes and operators relevant to that attach point.

IS-IS supports routing policy attachment at specific points that control route origination and the propagation of routes between levels:

- default-information originate
- inter-area-propagate

default-information originate

The default-information originate attach point conditionally injects the default route `0.0.0.0/0` into the IS-IS route database.

In this example, the policy injects an IPv4 unicast default route when the local RIB contains any route that matches *10.0.0.0/8 ge 8 le 25*. The policy sets the default route metric to *100* and the level to *level-1-2*.

Example:

```
route-policy isis-originate
  if rib-has-route in (10.0.0.0/8 ge 8 le 25) then
    set metric 100
    set level level-1-2
  endif
end-policy


router isis instance_10
  address-family ipv4 unicast
    default-information originate policy isis-originate
```

inter-area-propagate

The inter-area-propagate attach point conditionally propagates prefixes between IS-IS levels within the same IS-IS instance.

In this example, the policy allows prefixes to propagate from level 1 to level 2 when the destination matches *10.0.0.0/8 ge 8 le 25*.

Example:

```
route-policy isis-propagate
  if destination in (10.0.0.0/8 ge 8 le 25) then
    pass
  endif
end-policy


router isis instance_10
  address-family ipv4 unicast
    propagate level 1 into level 2 policy isis-propagate
```

Modify routing policies

Explains how to update attached and unattached policies safely, preserve references, and use nondestructive editing behavior to avoid unintended policy replacement.

Modifying routing policies is the process of changing route policies that

- updates an attached policy by respecifying it at the attach point to avoid a gap where default behavior applies
- allows a nonattached policy to reference sets or policies that you define later, while requiring all references to exist when you attach the policy, and

- supports nondestructive edits by using **rpl set-exit-as-abort** so **exit** aborts changes instead of applying and replacing the existing policy.

Attached policy modification

Describes how to update a policy already used at an attach point by respecifying it there to avoid a gap that would trigger default behavior.

You may need to change a policy that is already used at an attach point. If you remove and create the policy again, there may be a time when the attach point has no policy. During this gap, the system uses the default behavior.

To avoid this gap, set the policy again at the attach point. This process ensures that the system updates the policy without any time when no policy is applied.

> **Note**
>
> You cannot remove a route policy or set that is used at an attach point. The system does not remove the policy and shows an error because removing the policy would create an undefined reference.

Nonattached policy modification

Explains how unattached policies can reference undefined sets or policies temporarily, while requiring all references to exist before attachment.

If you do not attach a policy to an attach point, you can reference sets or policies that do not exist yet. Build configurations in stages and define any missing sets or policies later.

For example, a policy can use **apply** to reference another policy that you plan to create later, and a policy statement can reference a set that is not defined yet.

When you attach a policy, ensure that all referenced policies and sets exist. If you attach a policy that references an undefined policy or set, the system rejects the configuration.

Nondestructive editing of routing policy

Describes how nondestructive editing changes exit behavior so edits can be aborted instead of applied when working in routing policy configuration mode.

Nondestructive editing changes the behavior of the **exit** command in routing policy configuration mode. With nondestructive editing, **exit** aborts the edit rather than applying it.

By default, the **exit** command acts as **end-policy**, **end-set**, or **end-if**. When you use **exit** in routing policy configuration mode, the system applies your changes. This updates the configuration and replaces the existing policy.

Use **rpl set-exit-as-abort** command to override the default behavior of **exit** command in routing policy configuration mode.

Edit routing policy configuration elements

Teaches you how to update RPL statements and set content by using the CLI or supported editors while controlling whether changes are committed or discarded.

Editing routing policy configuration elements is the process of changing RPL policy content that

- treats policies as statement-based, where new lines act only as separators inside the begin-end policy structure
- lets you add or delete route policy statements by using the CLI, and
- lets you edit policy content between the begin-end brackets by using supported text editors such as Nano and Emacs.

Edit routing policy configuration elements using CLI

Teaches you how to add, delete, complete, or discard route policy statements from the CLI while editing a policy or set.

Add, delete, or discard route policy statements by using the CLI.

1. Enter the routing policy configuration mode for the policy or set you want to edit.
2. Enter route policy statements to add content, or delete existing statements as needed.
3. Complete the configuration block using one of these commands.
   - **end-policy** or **end-set**
   - **exit**
4. To discard your changes, use the **abort** command to exit configuration mode without applying changes.

The system updates the policy or set when you complete the configuration block, or discards your changes when you use **abort**.

Edit routing policy configuration elements using Emacs editor

Teaches you how to edit a route policy in Emacs, save or correct changes, and commit updates only when the policy parses successfully.

Edit your route policy in Emacs text editor and commit the changes from the editor.

Follow these steps to edit a routing policy by using Emacs editor:

1. In EXEC mode, enter **edit route-policy** <**name**>.

```
Router# edit route-policy policy_A
```

   The system copies your route policy to a temporary file and launches the editor.

2. In the editor, update your policy text as needed.
3. After editing, save your changes and remain in the editor, or save your changes and exit the editor.
   - To save the changes and remain in the editor, press **Ctrl-X**, and then **Ctrl-S**.
   - To save the changes and exit the editor, press **Ctrl-X**, then **Ctrl-C**.
4. If the system prompts **Continue editing? [no]:** because of parse errors, you can::
   - Enter **yes** to correct the text.
   - Enter **no** to end your session without changing the running configuration.

   Edit and commit a policy with no parse errors

```
Router# edit route-policy policy_A

== MicroEMACS 3.8b () == rpl_edit.139281 ==
  if destination in (2001::/8) then
    drop
  endif
end-policy
!


== MicroEMACS 3.8b () == rpl_edit.139281 ==
Parsing.
83 bytes parsed in 1 sec (82)bytes/sec
Committing.
1 items committed in 1 sec (0)items/sec
Updating.
Updated Commit database in 1 sec
```

Handle parse errors

```
Router# edit route-policy policy_B
== MicroEMACS 3.8b () == rpl_edit.141738
route-policy policy_B
  set metric-type type_1
  if destination in (2001::/8) then
    drop
  endif
end-policy
!
== MicroEMACS 3.8b () == rpl_edit.141738 ==
Parsing.
105 bytes parsed in 1 sec (103)bytes/sec


% Syntax/Authorization errors in one or more commands.!!
CONFIGURATION FAILED DUE TO SYNTAX/AUTHORIZATION ERRORS
  set metric-type type_1
  if destination in (2001::/8) then
    drop
  endif
end-policy
!


Continue editing? [no]:
```

If your policy has no parse errors, the system commits your updated policy. If you exit and do not continue after correcting parse errors, the system does not change the running configuration.

Edit routing policy configuration elements using Vim editor

Teaches you how to open, modify, save, or discard route policy changes by using the Vim editor.

Edit a routing policy by using the Vim editor.

1. Open the routing policy in Vim.

```
Router# edit route-policy test vim
```

2. Edit the policy content as needed.
3. When you complete changing the policy content, you can:
   - Save your changes and exit Vim by using **:wq**, **:x**, or **ZZ**.
   - Quit Vim editor by using **:q**. The system prompts whether to save the changed configuration.
   - Quit Vim without saving changes by using **:q!**.

The system uses the saved edits when you exit after saving. The system discards the edits when you quit without saving.

For more information, see http://www.vim.org/.

Edit routing policy configuration elements using Nano editor

Teaches you how to edit a route policy in Nano and commit the saved changes when you exit the editor.

Edit a route policy in the Nano editor.

1. In EXEC mode, enter the **edit route-policy <name> nano** command.

```
Router# edit route-policy test nano
```

The system launches the Nano editor.

2. In the Nano editor, edit the policy content as needed.
3. Press `Ctrl-X` to save the file and exit the editor.

The system commits the saved changes to the route policy.

For more information, see http://www.nano-editor.org/.

> **Note**
>
> The system does not support all Nano editor features.

Edit routing policy language set elements using XML

Teaches you how to update set elements through XML by appending, prepending, deleting, and committing entries without replacing the entire set.

Follow these steps to edit set elements by using XML:

1. Open the XML representation for the set you want to update.
2. Append entries to add items to the end of the set.
3. Prepend entries to add items to the beginning of the set.
4. Delete entries to remove items from the set.
5. Commit the XML changes.

The existing set is updated with new or removed entries without replacing the entire set.

6 Policy Based Routing

Topics:

Describes how policy-based routing enables routing decisions based on parameters beyond destination IP address, allowing granular traffic control and optimization.

Policy-based routing

Explains the benefits of policy-based routing by allowing administrators to define routing policies using parameters such as IP address, port number, or protocol. It supports granular traffic control and optimization through features like flow-tag and forward-class.

Policy-based routing is a routing technique that

- lets administrators define routing policies by using parameters such as IP address, port number, or protocol
- makes routing decisions based on criteria other than the destination IP address, and
- supports granular traffic control and optimization by using features such as flow-tag, forward-class, and access-group.

Table 32: Feature History Table

Feature Name	Release Information	Feature Description
Policy-Based Routing byte count	Release 26.1.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100, P200], 8700 [ASIC: P100, K100], 8010 [ASIC: A100])(select variants only); Modular Systems (8800 [LC ASIC: P100]) (select variants only*) The bytes count enables the router to accurately track and display the total number of bytes matched by each policy-based routing (PBR) rule. This provides enhanced visibility into traffic volume, allowing you to better monitor and analyze network usage. The byte count in the **show pbr stats counters policy <policy_name>** command now supports additional PIDs. *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 8712-MOD-M • 8011-4G24Y4H-I • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM • 8223-64EF-M(O)
Policy-Based Routing	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100]) (select variants only*) *This feature is supported on: • 8011-32Y8L2H2FH • 8011-12G12X4Y-A • 8011-12G12X4Y-D
Policy-Based Routing	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100]) (select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.

Feature Name	Release Information	Feature Description
Policy-Based Routing	Release 24.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100]) (select variants only*) *This feature is supported on Cisco 8712-MOD-M routers.
Policy-Based Routing	Release 24.2.11	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100]) (select variants only*); Modular Systems (8800 [LC ASIC: P100]) (select variants only*) You can now create customized routing policies based on different parameters such as IP address, port numbers, or protocols. With policy-based routing (PBR), you can enhance your network security by steering sensitive data away from potentially vulnerable network segments. Also, by allowing you to distribute traffic across multiple paths, PBR can help prevent traffic congestion in your network. *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Policy-based routing (PBR) lets you route packets by configuring a policy for traffic flows. This reduces reliance on routing protocols. PBR allows you to prioritize and provide a specific routing path for certain types of traffic, such as VoIP and video conferencing, based on the service-level agreement (SLA).

Unlike traditional routing, which relies solely on the destination IP address, PBR enables you to route packets based on various parameters, such as IP address, port numbers, or protocols. For instance, you can implement routing policies to permit or deny traffic paths based on the identity of a specific end system, an application protocol, or packet size.

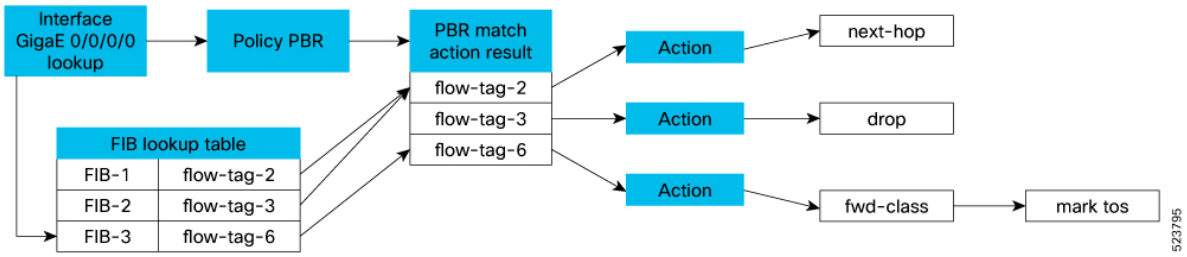
Flow-tag support

A provider edge (PE) router directs traffic toward the core through learned routes and installs policies on your interfaces based on your profiles. You must manually scan route tables and install Access Control Lists (ACLs) on your interfaces for proper traffic forwarding. As the scale of route prefix lists increases, the hardware resources required to program policies also increase. Whenever route updates occur, you must manually update these policies. You can apply route policies per prefix list but cannot select them for each interface to associate policies for each customer. This limitation may require additional planning.

To simplify policy management and improve scalability, you can classify the traffic early in the routing domain as routes are learned, and mark them with metadata or a tag in the Forwarding Information Base (FIB). Forwarding-path rules can be defined against this flow-tag value, reducing manual updates and improving scalability.

The system sets and distributes the flow tag via the Routing Information Base (RIB) as a policy attribute of the Forwarding Information Base (FIB) entry in the FIB lookup table. Use the Route Policy Language (RPL) set operation to create flow tags. The PBR policy then refers to the flow tag and associates it with specific actions or policy rules for its value.

Figure 1: Flow-tag process



Forward-class support

Use the PBR class-map to define matching criteria for a particular type of traffic. Use the forward-class to specify the forwarding path for packets. After you associate a class-map with a forwarding-class in the policy map, all matching packets are forwarded as specified in the policy map.

Use Traffic Engineering (TE) tunnels to direct traffic along specific network paths. Associate TE tunnels with a forward-class to determine the forwarding path for packets.

The use of the **auto-route** command allows the TE interface to be exported to the routing protocol module, associating the route in the Forwarding Information Base (FIB) database with these tunnels.

The system can support up to eight forward-classes with eight TE tunnels each, allowing a maximum of 32 TE tunnels to be associated with the destination route.

Guideline for forward-class implementation

When you configure `match tcp-flag`, also configure `match protocol tcp` to ensure correct traffic classification.

Access-group support

Permit or deny actions in ACL groups

PBR supports matching on an ACL group ID. You can specify large prefix lists and long lists of specific permit or deny lists. The prefix lists filter routing updates, while the permit or deny lists determine the action taken when a match occurs.

- Permit - Stop processing ACL, and if
  - the type of class is “match-any”, proceed to take policy-map action
  - the type of class is “match-all”, proceed to the next ACL or “match” statement in the class-map considering that you already have a match (true) for an existing ACL.
- Deny - Stop processing ACL and if
  - the type of class is “match-any”, proceed to the next ACL (or “match”) statement in the class-map if there is no match for the existing ACL.
  - the type of class is “match-all”, exit the class. No policy action is taken, and the next class is processed as per the specified order.

When the ACE is a MISS, that is, it did not match any permit or deny statement, these actions are performed:

- Proceed to the next "match" statement.
- If the “match” statement does not provide a match, then skip that class. No policy action is taken for that class, and proceed to check the packet against the next class in order, or proceed to L3 forwarding lookup.

> **Note**
>
> An explicit deny entry is not supported in ACL groups that are embedded in class-maps.

Policy-based routing example

For example, you can implement a policy to route VoIP traffic through a dedicated path to ensure quality of service, while video conferencing traffic can be prioritized using a specific SLA. PBR can also be used to steer sensitive data away from vulnerable network segments, and distribute traffic across multiple paths to prevent congestion.

Supported match and set operations

Describes the criteria supported for matching and setting in policy-based routing, including source and destination IP, protocol, ports, access groups, flow-tags, and nexthop settings.

Supported match and set operations in PBR refer to the criteria that can be used to identify and manipulate network traffic for routing purposes.

- Match operations include source IP, destination IP, source protocol/port, destination protocol/port, access-group, flow-tag, IP protocol, TCP flag, and port-range.
- Set operations include nexthop IP, nexthop VRF, nexthop IP+VRF, and forward-class.

Table 33: Supported match and set operations

Criteria	match/set
source ip	match
destination ip	match
source protocol/port	match
destination protocol/port	match
nexthop ip	set
nexthop vrf	set
nexthop ip+vrf	set
forward-class	set
access-group	match
flow-tag	match
ip protocol	match
tcp-flag	match
port-range	match

Restrictions for implementing policy-based routing

Provides guidelines and limitations for implementing policy-based routing, such as incompatibilities between QoS group and flow-tag features and scale considerations..

These restrictions apply when implementing Policy-based routing.

- QoS Group and Flow-tag are not supported together at the same time.
- Bridge Group Virtual Interface (BVI) and Pseudowire Headend (PWHE) subinterfaces support PBR from Release 26.1.1.
- BGP Flowspec feature and PBR are not supported together on the same interface.
- A route-policy can have either 'set qos-group' or 'set flow-tag,' but not both for a prefix-set.
- Route policy for qos-group and route policy flow-tag cannot have overlapping routes. The Quality-of-service Policy Propagation Using Border Gateway Protocol (QPPB) and flow tag features can coexist (on same as well as on different interfaces) as long as the route policy used by them do not have any overlapping route.
- Mixing usage of qos-group and flow-tag in route-policy and policy-map is not recommended.

Configure policy-based routing

Configure policy-based routing (PBR) by setting up flow-tags, provisioning forward classes using RPL, and configuring ACLs with policy-based routing.

1. Configure flow-tag.
2. Provision forward class using RPL.
3. Configure ACLs with policy-based routing.

Configure the flow-tag

Configure the Flow-tag in route policies, apply the policy to the routing table, and use class maps and policy maps to manage traffic flows on interfaces.

Configure the Flow-tag to classify and manage traffic flows using route policies, class maps, and policy maps.

The Flow-tag feature allows you to mark and control traffic flows for advanced routing and policy-based forwarding. This procedure demonstrates how to define AS path sets, set Flow-tags, and apply related configurations.

1. Define a named AS path set in the route policy.

```
Router(config)#as-path-set as-set-1
   ios-regex '_12$',
   ios-regex '_13$'
end-set
```

   This AS path set is referenced in route-policy configuration to match specific AS paths.

2. Set the Flow-tag under route-policy configuration.

```
Router(config)# route-policy flowtag_match
Router(config-rpl)# if community matches-every (100:1) then
set flow-tag 31
else
Router(config-rpl-else)# if as-path in as-set-1 then
set flow-tag 62
Router(config-rpl-else)# endif
Router(config-rpl)# end-policy
```

   The route-policy uses community and AS path matches to set different Flow-tags.

3. Apply the policy when updating the routing table.

```
Router(config)# router bgp 100
Router(config-bgp)# bgp router-id 209.165.201.19
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# table-policy flowtag_match
```

   Ensure the policy is applied to the correct BGP address-family for traffic classification.

4. Configure a class map and policy map for traffic matching and forwarding.

```
Router(config)# class-map type traffic match-all green-tag1
Router(config-cmap)# match flow-tag 31
Router(config-cmap)# end-class-map
Router(config-cmap)# exit
Router(config)# policy-map type pbr nh_select
Router(config-pmap)# class type traffic green-tag1
Router(config-pmap-c)# set forward-class 1
```

   The class map matches traffic with Flow-tag 31, and the policy map sets the forwarding class for matched traffic.

5. Configure an interface and apply the PBR policy map to the interface.

```
Router(config)# interface HundredGigE0/7/0/27
Router(config-if)# ipv4 address 10.10.20.10
Router(config-if)# service-policy type pbr input nh_select
```

   Applying the policy map to the interface enables policy-based routing for incoming traffic.

6. Verify the running configuration for class map, policy map, and interface.

```
Router# show running-config class-map
class-map type traffic match-all green-tag1
match flow-tag 101
 end-class-map
!
Router# show running-config policy-map
policy-map type pbr nh_select
class type traffic green-tag1
  redirect ipv4 nexthop 10.1.2.2
 !
 class type traffic green-tag1
 !
 end-policy-map
!
Router# show running-config interface HundredGigE0/7/0/27
interface HundredGigE0/7/0/27
service-policy type pbr input nh_select
!
```

   Use these commands to confirm that Flow-tag, class map, policy map, and interface configurations are active.

Flow-tag configuration is applied, and traffic is classified and forwarded according to the defined policies.

Provision forward class using RPL

Explains how to provision Forward Class using Route Policy Language (RPL) in BGP, including configuration steps for community strings, VRFs, and next-hop policies to set the appropriate forward-class for traffic engineering tunnels.

Provisioning forward class using RPL involves configuring route policies that set the forward-class ID based on matching criteria such as community strings, VRFs, or next-hop addresses. This enables differentiated forwarding for traffic engineering tunnels.

- Route policies can match on community strings, VRFs, or next-hop sets.
- Forward-class IDs are set within the route policy and applied to BGP routes.
- Traffic is forwarded through TE tunnels associated with the configured forward-class.

Use this sample configuration to provision forward class using RPL.

1. Set the forward class ID for community string.

```
Router(config)# route-policy c1
Router(config-rpl)# if community matches-every (6500:1) then
set forward-class 1
Router(config-rpl-else)# endif
Router(config-rpl)# end-policy
Router(config)# router bgp 50
Router(config-bgp)# bgp router-id 209.165.201.29
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# table-policy c1
Router(config-bgp-af)# exit
Router(config-bgp)# exit
Router(config)# interface tunnel-te1
Router(config-if)# forward-class 1
```

2. Set the forward class ID for VRF.

```
Router(config)# route-policy c1
Router(config-rpl)# set forward-class 1
Router(config-rpl)# end-policy
Router(config)# route-policy c2
Router(config-rpl)# set forward-class 2
Router(config-rpl)# end-policy
Router(config)# router bgp 50
Router(config-bgp)# bgp router-id 209.165.201.29
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# exit
Router(config-bgp)# exit
Router(config-bgp)# vrf one
Router(config-bgp-vrf)# rd 1:1
Router(config-bgp-vrf)# address-family ipv4 unicast
Router(config-bgp-af)# table-policy c1
Router(config-bgp-af)# exit
Router(config-bgp)# exit
Router(config-bgp)# vrf two
Router(config-bgp-vrf)# rd 2:2
Router(config-bgp-vrf)# address-family ipv4 unicast
Router(config-bgp-af)# table-policy c2
Router(config-bgp-af)# exit
Router(config-bgp)# exit
Router(config)# interface tunnel-te1
Router(config-if)# forward-class 1
Router(config)# interface tunnel-te2
Router(config-if)# forward-class 2
```

3. Configure a route policy with next hop and set a forward class.

```
Router(config)# prefix-set nh-set-1
Router(config-pfx)# 10.10.0.1
Router(config-pfx)# end-set
```

```
Router(config)# route-policy c1
Router(config-rpl)# if next-hop in nh-set-1 then
set forward-class 1
Router(config-rpl)# endif
Router(config-rpl)# end-policy
Router(config)# router bgp 50
Router(config-bgp)# bgp router-id 209.165.201.29
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# table-policy c1
Router(config-bgp-af)# exit
Router(config-bgp)# exit
Router(config)# interface tunnel-te1
Router(config-if)# forward-class 1
```

In these examples, BGP on the receiving PE is configured with table policies that set the forward-class based on the matching criteria. The appropriate TE tunnel is selected for forwarding based on the forward-class ID.

For example, if a route matches community 6500:1, the route-policy sets forward-class 1, and traffic is forwarded through tunnel-te1. For VRF one and two, table-policies C1 and C2 set forward-class 1 and 2, selecting tunnel-te1 and tunnel-te2, respectively. When matching on next-hop 10.10.0.1, the policy sets forward-class 1 and selects the corresponding tunnel.

Configure ACLs with policy-based routing

Configures access control lists (ACLs) with policy-based routing (PBR) using sample configuration commands and verify the configuration on your router.

Use this procedure to configure ACLs with PBR. The following steps provide a sample configuration and verification commands.

1. Configure an access list.

```
Router(config)# ipv4 access-list INBOUND-ACL
Router(config-ipv4-acl)# 10 permit ipv4 any host 10.1.1.10
Router(config-ipv4-acl)# 20 permit ipv4 any host 10.2.3.4
Router(config-ipv4-acl)# commit
Router(config-ipv4-acl)# exit
```

2. Configure a class map for the access list.

```
Router(config)# class-map type traffic match-any INBOUND-CLASS
Router(config-cmap)# match access-group ipv4 INBOUND-ACL
Router(config-cmap)# end-class-map
Router(config)# commit
```

3. Configure a PBR policy map with the class map.

```
Router(config)# policy-map type pbr INBOUND-POLICY
Router(config-pmap)# class type traffic INBOUND-CLASS
Router(config-pmap-c)# redirect ipv4 nexthop 192.168.10.1
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic class-default
Router(config-pmap-c)# transmit
```

```
Router(config-pmap-c)# commit
Router(config-pmap)# end-policy-map
```

4. Configure a Gigabit Ethernet interface and apply the PBR policy map to the interface.

```
Router(config)# interface GigabitEthernet 0/0/0/0
Router(config-if)# ipv4 address 10.10.10.1 255.255.255.0
Router(config-if)# service-policy type pbr input INBOUND-POLICY
Router(config-if)# commit
Router(config-if)# exit
```

5. Verify the running configuration for the access list.

```
Router# show running-config ip access-list
ipv4 access-list INBOUND-ACL
10 permit ipv4 host 10.10.10.1 any
!
```

ePBR drop and transmit actions

Describes how enhanced policy-based routing introduces drop and transmit actions to control packet handling precisely, improving security and simplifying policy design.

ePBR drop and transmit actions is a routing feature that

- provides granular control over how packets matching specific criteria in a PBR policy are handled
- discards unwanted or malicious traffic directly at the interface to improve security and filtering
- transmits selected traffic, bypassing PBR rules and following the standard routing table lookup, and
- simplifies policy design and improves operational clarity with dedicated actions for common traffic management needs.

Table 34: Feature History Table

Feature Name	Release Information	Feature Description
ePBR drop and transmit actions	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is supported on Cisco 8711-48Z-M routers.

Feature Name	Release Information	Feature Description
ePBR drop and transmit actions	Release 25.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100], 8010 [ASIC: A100]); Modular Systems (8800 [LC ASIC: P100]) The feature adds two critical forwarding actions, such as `drop` and `transmit` to enhanced Policy-Based Routing (ePBR) policies, giving network administrators precise control over how traffic matching specific criteria in a PBR policy is handled. These actions simplify policy creation, remove complex workarounds, and allow administrators to manage exceptions or security scenarios with granular per-traffic control. CLI: • The **drop** and **transmit** commands are introduced in the policy-map.

Drop and transmit actions for enhanced PBR control

The feature enhances ePBR by introducing two new forwarding actions, such as drop and transmit. These actions give network administrators explicit control over the final disposition of traffic matching specific PBR criteria.

Drop action: Discards unwanted or malicious traffic directly at the ingress interface, such as malicious traffic flows, unauthorized protocols, or unwanted sources directly at the ingress interface. This early filtering removes the need for complex ACL configurations, and prevents harmful traffic from consuming network resources.

Transmit action: Forwards selected traffic normally, bypassing PBR-specific next-hop or policy redirection rules, and follows the standard FIB lookup. This ensures that the packet follows the standard routing table lookup and is treated like regular traffic.

These actions complement the existing `redirect` and `set forward-class` actions, providing more comprehensive and flexible traffic steering capabilities. By making these actions explicit within ePBR policies, network administrators avoid relying on implicit defaults, complex null interface configurations, or intricate policy ordering.

This capability delivers precise per-traffic handling, strengthens network security, preserves service integrity for critical flows, clarifies operational intent, reduces configuration errors, and streamlines policy design.

Benefits of ePBR drop and transmit actions

ePBR drop and transmit actions offer these benefits:

- Ensures granular security and filtering for precise traffic handling decisions.
- Enhances network security by dropping malicious or irrelevant packets early.
- Flexible exception handling through the transmit option.
- Simplifies ePBR policies.

Enable drop or transmit actions for PBR policies

Teaches you how to configure class maps and policy maps to assign drop or transmit actions for traffic classes, and how to apply these policies to interfaces.

Follow these steps to enable drop or transmit actions for PBR policies.

1. Run the **class-map type** command to define class maps for the traffic you want to control.

   Class maps define the match criteria for the PBR policy.

```
/* Configure class map to drop traffic */
Router(config)#class-map type traffic match-all UNWANTED-SOURCE-TRAFFIC
Router(config-cmap)#match source-address ipv4 192.168.1.0/24
Router(config-cmap)#end-class-map
```

```
/* Configure class map to transmit traffic */
Router(config)#class-map type traffic match-all MANAGEMENT-TRAFFIC
Router(config-cmap)#match destination-port 22
Router(config-cmap)#match destination-port 23
Router(config-cmap)#end-class-map
```

2. Run the **policy-map type** command to create a policy map of type PBR, and assign the new actions to the defined classes.

```
Router(config)#
policy-map type pbr INGRESS-SECURITY-POLICY
/* Assign the drop action to the UNWANTED-SOURCE-TRAFFIC class */
Router(config-pmap)#class type traffic UNWANTED-SOURCE-TRAFFIC
Router(config-pmap-c)#drop
Router(config-pmap-c)#exit
```

```
/* Assign the transmit action to the MANAGEMENT-TRAFFIC class */
Router(config-pmap)#class type traffic MANAGEMENT-TRAFFIC
Router(config-pmap-c)#transmit
Router(config-pmap-c)#exit
```

3. (Optional) Run the **class type** command to define a class default action when no other class matches.

   You can define a default action when traffic does not match any class, and it then follows normal routing.

```
Router(config-pmap)#class type traffic class-default
Router(config-pmap-c)#transmit
Router(config-pmap-c)#commit
Router(config-pmap)#end-policy-map
```

4. Apply the defined policy map to the required interface.

```
Router(config)#interface HundredGigE 0/0/0/1
Router(config-if)#service-policy type pbr input INGRESS-SECURITY-POLICY
Router(config-if)#commit
Router(config-if)#exit
```

5. Run the **show running-config** command to verify the configuration.

```
class-map type traffic match-all UNWANTED-SOURCE-TRAFFIC
 match source-address ipv4 192.168.1.0/24
end-class-map
```

```
class-map type traffic match-all MANAGEMENT-TRAFFIC
 match destination-port 22
 match destination-port 23
end-class-map
!
policy-map type pbr INGRESS-SECURITY-POLICY
 class type traffic UNWANTED-SOURCE-TRAFFIC
  drop
 !
 class type traffic MANAGEMENT-TRAFFIC
  transmit
 !
 class type traffic class-default
  transmit
 !
end-policy-map
!
interface HundredGigE 0/0/0/1
 service-policy type pbr input INGRESS-SECURITY-POLICY
!
```

6. Run the **show controllers npu stats** command to verify the dropped packets.

   In the sample output, the number of packets dropped is 9000.

```
Router#show controllers npu stats traps-all instance all location all | exclude
 "0                    0"
Tue Apr 15 19:44:11.681 PDT
…
Trap Type                                        NPU  Trap  Punt        Punt
Punt  Punt Configured Hardware   Policer Avg-Pkt Packets              Packets

                                                 ID   ID    Dest        VoQ
VLAN  TC  Rate(pps)  Rate(pps)  Level   Size    Accepted              Dropped
================================================================================
ONLINE_DIAG                                      0    32   RPLC_CPU    295
1538  7    8038660    8087158    IFG     64      362                   0


                              L3_ACL_DROP(D)

                                                                0    106  NPU_DROP
 ---   ---   ---  ---------- ---------- ---     N/A     0


                                   9000

ISIS/L3                                          0    133  RPLC_CPU    295
1538  7    10000      9789       IFG     1520    14                    0
```

ePBR on BVI

Explains how enhanced policy-based routing on Bridge Virtual Interfaces applies ingress security policies and traffic steering to Layer 2 traffic before Layer 3 routing.

Enhanced Policy-Based Routing (ePBR) on Bridge Virtual Interface (BVI) is a routing feature that

- applies ingress security policies and traffic steering to Layer 2 traffic entering a Provider Edge (PE) router through a BVI
- processes packets before they are routed to a Layer 3 interface, and
- provides granular control over traffic handling through dedicated redirect, drop, and transmit actions.

Table 35: Feature History Table

Feature Name	Release Information	Feature Description
ePBR on BVI	Release 26.1.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100]) (select variants only*); Centralized Systems (8400 [ASIC: K100]) (select variants only*); Modular Systems (8800 [LC ASIC: P100]) (select variants only*) You can ensure secure and efficient traffic handling at the network ingress by applying ePBR policies directly to the BVI. This feature allows the Cisco IOS XR software to intercept and steer inbound Layer 2 traffic before it transitions to Layer 3 routing. *This feature is supported on: • 8212-48FH-M • 8404-SYS-D • 8711-32FH-M • 8711-48Z-M • 8712-MOD-M • 88-LC1-36EH • 88-LC1-52Y8H-EM • 88-LC1-12TH24FH-E

The ePBR on BVI feature allows you to apply ingress security policies and traffic steering to Layer 2 traffic entering a PE router from a VPN through a BVI before it is routed to a Layer 3 interface.

By applying ePBR policies to a BVI, you can:

- Redirect: Forward traffic to a specific next-hop, bypassing the standard routing table.
- Drop: Discard malicious or unauthorized traffic at the ingress interface.
- Transmit: Explicitly permit traffic to follow standard routing table lookups.

For more information on ePRB, see ePBR drop and transmit actions on page 106.

Limitations for ePBR on BVI

Use these guidelines when configuring ePBR on BVI interfaces to ensure compliance with supported directions and scale limits.

These are the limitations for ePBR on BVI. For more restrictions about policy-based routing, see Restrictions for implementing policy-based routing on page 102.

- The policy supports only the ingress direction on the BVI.

- Supports up to 6k scale limits.

How ePBR on BVI works

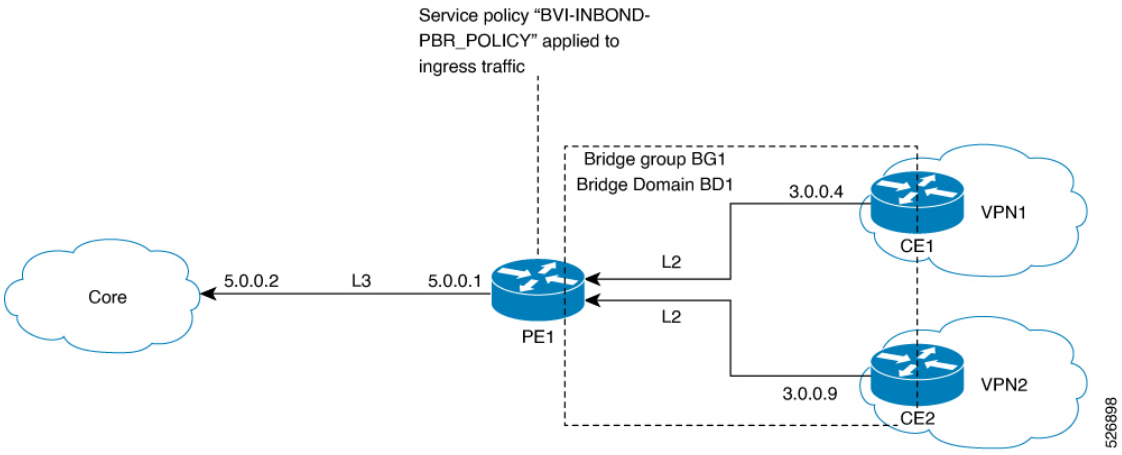
Describes how ePBR is applied to a BVI interface in a PE environment to steer and filter inbound Layer 2 traffic from customer VPNs before Layer 3 routing.

In a PE environment, managing Layer 2 traffic from customer VPNs requires granular control before the traffic transitions to Layer 3 routing. Applying ePBR directly to the BVI allows the system to intercept and steer this traffic based on the requirements.

Consider a topology where the PE router connects CE devices to the core network using a BVI as the Layer 3 gateway.

- **CE1 and CE2**: Customer edge devices.
  - **CE1**: Source router in VPN1 with IP address 3.0.0.4.
  - **CE2**: Source router in VPN2 with IP address 3.0.0.9.
- **PE1**: Provider edge router.
- **Core**: Core router with IP address 5.0.0.2.
- **BVI20**: The routed interface with IP address 3.0.0.8 on PE1 that acts as the gateway for Bridge Group BG1 and Bridge Domain BD1.
- **BVI-INBOUND-PBR_POLICY**: The service policy "BVI-INBOUND-PBR_POLICY" is configured on the BVI20 interface on PE1 to define specific actions, such as transmit, drop, and redirect, which manage inbound traffic from the CEs before it reaches the core router.

Figure 2: Network topology for ePBR on BVI



These stages describe how Cisco IOS XR software manages inbound traffic steering and filtering on a BVI using ePBR:

1. The CE devices CE1 or CE2 send an inbound packet containing Ethernet, VLAN, and IP headers to the PE1 router through the Layer 2 bridge domain.
2. The Cisco IOS XR software receives the packet and directs it to the BVI20 interface for Layer 3 processing.
3. The Cisco IOS XR software evaluates the packet against the **BVI-INBOUND-PBR_POLICY** attached to the BVI.
4. Based on the policy match, the Cisco IOS XR software performs one of the following actions:

If the packet matches the...	Then the Cisco IOS XR software...
transmit class	transmits the packet normally using standard routing lookups.

If the packet matches the...	Then the Cisco IOS XR software...
drop class	discards the packet at the ingress interface.
redirect class	steers the packet to the core next-hop at IP 5.0.0.2.

5. If the packet does not match any specific class, the Cisco IOS XR software applies the **class-default** action and forwards the traffic according to the standard routing table.

The ePBR process results in the automated filtering and steering of inbound Layer 2 traffic, ensuring that packets are handled according to security and traffic engineering policies before they reach the network core.

Configure ePBR on BVI

Teaches you how to configure BVI interfaces, bridge domains, pseudowire connections, and apply ePBR policies to enforce traffic steering and security rules on inbound Layer 2 traffic.

Perform the following steps to configure ePBR on BVI:

1. Configure the BVI interface on the local PE1 to establish the Layer 3 gateway for customer VPN traffic.

```
Router# configure
Router(config)# interface BVI20
Router(config-if)# ipv4 address 3.0.0.8 255.255.255.0
Router(config-if)# commit
```

2. Configure the bridge domain and associate physical interfaces with the BVI to enable routing between the Layer 2 domain and the Layer 3 network.

```
Router# configure
Router(config)# l2vpn
Router(config-l2vpn)# bridge group bg1
Router(config-l2vpn-bg)# bridge-domain bd1
Router(config-l2vpn-bg-bd)# interface FourhundredGigE0/1/0/27
Router(config-l2vpn-bg-bd-ac)# interface FourhundredGigE0/0/0/8
Router(config-l2vpn-bg-bd-ac)# routed interface BVI20
Router(config-l2vpn-bg-bd-bvi)# commit
```

3. Configure a point-to-point pseudowire connection with an xconnect group and neighbor specification on remote PE2 to establish a seamless Layer 2 VPN link between local PE1 and remote PE2.

```
Router# configure
Router(config)# l2vpn
Router(config-l2vpn)# xconnect group phy
Router(config-l2vpn-xc)# p2p p1
Router(config-l2vpn-xc-p2p)# interface FourhundredGigE0/0/0/7
Router(config-l2vpn-xc-p2p)# interface FourhundredGigE0/0/0/11.1
Router(config-l2vpn-xc-p2p)# commit
```

4. Configure the traffic class or classmap for IPv4 and IPv6 to identify the traffic based on specific criteria.
   The following is an example of classmap for IPv4:

```
Router(config)# class-map type traffic match-all ipv4_CM3
Router(config-cmap)# match destination-address ipv4 201.0.3.1 255.255.255.0
Router(config-cmap)# match source-address ipv4 192.1.1.0 255.255.255.0
Router(config-cmap)# match protocol tcp
Router(config-cmap)# match destination-port 1024
Router(config-cmap)# match source-port 1024
```

```
Router(config-cmap)# match tcp-flag 0x10
Router(config-cmap)# match access-group ipv4 ipv4_acl_3
Router(config-cmap)# match flow-tag 20
Router(config-cmap)# end-class-map
```

The following is an example of classmap for IPv6:

```
Router(config)# class-map type traffic match-all ipv6_CM1
Router(config-cmap)# match destination-address ipv6 2001:0:0:1::1/64
Router(config-cmap)# match source-address ipv6 1111::1:0/120
Router(config-cmap)# match protocol tcp
Router(config-cmap)# match destination-port 1024
Router(config-cmap)# match source-port 1024
Router(config-cmap)# match tcp-flag 0x10
Router(config-cmap)# match access-group ipv6 ipv6_acl
Router(config-cmap)# match flow-tag 10
Router(config-cmap)# end-class-map
```

5. Create the ePBR policy-map for IPv4 and IPv6 to define the actions such as redirect, drop, or transmit for the classified traffic.
   The following is an example of ePBR policy-map for IPv4:

```
Router(config)# policy-map type pbr BVI-INBOUND-PBR_POLICY
Router(config-pmap)# class type traffic ipv4_CM1
Router(config-pmap-c)# redirect ipv4 nexthop 192.0.1.2
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic ipv4_CM2
Router(config-pmap-c)# drop
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic ipv4_CM3
Router(config-pmap-c)# transmit
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic class-default
Router(config-pmap-c)# transmit
Router(config-pmap-c)# commit
Router(config-pmap)# end-policy-map
```

   The following is an example of ePBR policy-map for IPv6:

```
Router(config)# policy-map type pbr BVI-INBOUND-PBR_POLICY
Router(config-pmap)# class type traffic ipv6_CM1
Router(config-pmap-c)# redirect ipv6 nexthop 2001:101::2
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic ipv6_CM2
Router(config-pmap-c)# drop
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic ipv6_CM3
Router(config-pmap-c)# transmit
Router(config-pmap-c)# exit
Router(config-pmap)# class type traffic class-default
Router(config-pmap-c)# transmit
Router(config-pmap-c)# commit
Router(config-pmap)# end-policy-map
```

6. Attach the policy to the BVI interface to enforce traffic steering and security rules on inbound packets.

```
Router# configure
Router(config)# interface BVI20
```

```
Router(config-if)# service-policy type pbr input BVI-INBOUND-PBR_POLICY
Router(config-if)# commit
```

7. Run these commands to verify the policies:

a) Run the **show interfaces BVI1 accounting** command to verify interface statistics.

```
Router# show interface BVI1 accounting
BVI1
  Protocol              Pkts In         Chars In     Pkts Out        Chars
 Out
  ARP                         0                0            2
   84
  IPV6_ND                     0                0           35
 3544
```

b) Run the **show controllers npu stats traps-all instance all location all** to verify the NPU drop counters.

```
Router# show controllers npu stats traps-all instance all location all
Trap Type                                       NPU  Trap  Punt       Punt
 Punt  Punt Configured Hardware   Policer Avg-Pkt Packets
Packets
                                                ID   ID    Dest       VoQ
 VLAN  TC   Rate(pps)  Rate(pps)  Level   Size    Accepted
Dropped
================================================================================
L3_ACL_DROP(D)                                   0    106  NPU_DROP    ---
 ---   ---  ---------- ---------- ---     N/A     0                      0
```

c) Run the **show ofa objects ip4pbr location** or **show ofa objects ip6pbr location** command to verify PBR programming.

The following example displays the **show ofa objects ip4pbr location** command output:

```
Router# show ofa objects ip4pbr location
ip4pbr element 0 (hdl:0x309826b098):
  base
  |-- dpd_slf -   pending(cr/up/dl):0/0/0, sibling:0x3094b99c78, child:0,
 num_parents:1, visits:0
   color_mask:0, last_bwalk_id:0 num_bwalks_started:0
  |-- flag - 4000
      |-- flag.is_fwalk_true - 0x1
  |-- keylen - 77
  |-- trans_id - 88551
  |-- create_trans_id - 88499
  |-- obj_handle - 0x309826b098
  |-- obj_rc - 0x0
  |-- reason - 0
  |-- table_operation - 6
  |-- total_obj_size - 600
  |-- idempotent - 0
  |-- inflight - 0
  |-- table_prop - jid=258 mtime=(UTC)2025.Oct.22 13:36:45.811468
  |--  (cont'd)  - replayed=0times
  `-- obj_rc - 0:Success
 ofa_npu_mask_t npu_mask => 1
@uint32_t npu_id => 0
@ofa_policymap_name_t policymap_name => BVI-INBOUND-PBR_POLICY:0
 uint32_t pbr_acl_id => 1
@uint32_t ace_seq_num => 0
@uint32_t entry_index => 0
```

```
  dpa_ip_addr_t src_ip_addr => 192.1.1.0
  dpa_ip_mask_t src_ip_mask => 255.255.255.0
  dpa_ip_addr_t dest_ip_addr => 201.0.1.1
  dpa_ip_mask_t dest_ip_mask => 255.255.255.0
  uint8_t tcp_flags => 16
  uint8_t tcp_flags_mask => 255
  dpa_l4_port_t src_port => 1024
  dpa_l4_port_mask_t src_port_mask => 65535
  dpa_l4_port_t dest_port => 1024
  dpa_l4_port_mask_t dest_port_mask => 65535
  dpa_port_range_info src_port_range => (not set)
  dpa_port_range_info dest_port_range =>     0/0
  uint8_t proto => 6
  uint8_t proto_mask => 255
  uint8_t flowtag => 10
  uint8_t flowtag_mask => 63
  uint8_t forward_class_id => (not set)
  uint8_t mark_dscp => (not set)
  uint8_t match_dscp => (not set)
  uint64_t transportnh_id => 1
  ofa_bool_t is_acl_delete => (not set)
  ofa_pbr_priority_t priority_str =>
  dpa_pbr_class_name_st class_name => (not set)
  uint64_t nhg_id_match => (not set)
  uint64_t gid_match => (not set)
  uint64_t redirect_nhg_id => (not set)
  uint64_t redirect_gid => (not set)
  uint64_t rule_id => (not set)
  ofa_bool_t last_entry => (not set)
  ofa_bool_t is_drop => (not set)
  ofa_bool_t is_transmit => (not set)
  dpa_npu_mask_t npu_bmap => 4
  dpa_transportnh_hdl_t transportnh_refhdl => (not set)
  transportnh_obj.refs_union => transportnh
  transportnh_obj.transportnh.refkey dpa_transportnh_id_t transportnh_id
=> (not set)
  transportnh_obj.transportnh.refhdl => 0x30983a0098
```

The above sample displays only a part of the actual output; the actual output displays more details.

7 Implementing Static Routes

Topics:

Explains how to configure and manage static routes, including route types, VRF use, multicast routing, cross address-family routing, scalability limits, and load-balancing behavior through configuration examples.

Static routes

Use this topic to learn about static route functional overview.

A static route is a manually configured route that

- defines an explicit path between networking devices without using dynamic routing protocols
- does not update automatically when the network topology changes, and
- reduces bandwidth and CPU use while providing controlled routing behavior.

Static routes are not automatically updated and must be manually reconfigured if the network topology changes. They can be redistributed into dynamic routing protocols, but dynamic routes cannot be redistributed into the static routing table.

Table 36: Feature History Table

Feature Name	Release Information	Feature Description
Conditional Static Anycast Route Advertisement	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is supported on Cisco 8711-48Z-M routers.
Conditional Static Anycast Route Advertisement	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.
Conditional Static Anycast Route Advertisement	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 8712-MOD-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Feature Name	Release Information	Feature Description
Conditional Static Anycast Route Advertisement	Release 7.3.15	This feature enables you to to detect core isolation and improve network convergence. The conditional installation of static route in routing table (RIB) based on object state functionality has been enhanced to use event driven notification, instead of the polling mechanism used earlier. This event driven approach allows faster detection and notification of object state changes. This feature along with BGP Neighbor address-family tracking, allows users to detect network faults quicker and take corrective action faster. This feature improves network convergence and reduces, and in some cases, eliminates traffic outages in the event of certain network faults, such as either link or node failures or both. Without this new feature, it will take longer to detect network faults and increase the duration of network outage.

Use static routes if your network has only one path to an outside network. You can also use static routes to control certain types of traffic or to secure specific links in a larger network. Most networks use dynamic routing protocols for communication, but you may configure one or two static routes for specific situations.

Prerequisites for implementing static routes

Lists the access and task group requirements needed to configure static routes and advises contacting the AAA administrator if permissions prevent command use.

You must be in a user group associated with a task group that includes the proper task IDs. The command reference guides include the task IDs required for each command. If you suspect user group assignment is preventing you from using a command, contact your AAA administrator for assistance.

Restrictions for implementing static routes

Provides restrictions and forwarding considerations for static routes, including next-hop configuration guidance and exceptions involving leaked prefixes.

If your next hop is part of a local subnet, configure static routes in the global table and specify the egress interface as the next hop. To prevent packet loss, configure static routes in the global table and set the next hop IP address as the next hop.

In most cases, a route is learned from the AIB in the global table and is installed in the FIB. This process does not apply to leaked prefixes, which may cause inconsistencies in forwarding.

Default administrative distance

Explains how administrative distance affects preference between static and dynamic routes and how a higher static-route distance allows dynamic routes to take precedence.

The default administrative distance is a route preference value that

- determines how the system prefers static routes relative to dynamic routes
- uses a default value of `1` for static routes, and
- allows dynamic routes to override static routes when you configure a higher administrative distance for the static route.

Default VRF

Describes how every static route must belong to a VRF and explains that routes use the default VRF when no other VRF is specified.

You must always associate a static route with a VPN routing and forwarding (VRF) instance. You can use the default VRF or specify another VRF. To configure a static route for a particular VRF, use the **vrf***vrf-name* command to enter VRF configuration mode. If you do not specify a VRF, the static route is configured for the default VRF.

> **Note**
>
> When you configure a static route for IPv4 or IPv6 in any VRF, it works the same as a static route in the default VRF. Each VRF supports both the IPv4 and IPv6 address families.

IPv4 and IPv6 static VRF routes

Describes how IPv4 and IPv6 static routes operate within a specific VRF in the same way they operate in the default VRF.

An IPv4 or IPv6 static VRF route is a static route configured within a specific VRF, operating in the same way as a static route for the default VRF.

Directly connected routes

Explains how directly connected static routes point to an output interface and become valid only when the interface is up and enabled for the address family.

A directly connected route is a static route that

- points only to an output interface
- is treated by the routing table as directly attached, and
- becomes a candidate for insertion only when the referenced interface is valid, up, and has IPv4 or IPv6 enabled.

Advertisement of directly connected routes

Directly connected routes are advertised by IGP routing protocols when the interface is configured for the protocol.

If you create a static route that points to an interface, the system considers it directly connected. The routing protocol treats the destination as directly attached to the specified interface.

Configure a directly connected static route

Teaches you how to configure a static route that reaches a destination prefix through a specified output interface.

1. In static route configuration mode, specify the address family.

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)#
```

2. Define the address prefix and output interface.

   This example shows how to specify that all destinations with address prefix 2001:0DB8::/32 are directly reachable through interface FourHundredGigE 0/0/0/0.

```
Router(config-static-afi)# 2001:0DB8::/32 FourHundredGigE 0/0/0/0
Router(config-static-afi)#commit
```

Fully specified static routes

Describes how fully specified static routes define both the outgoing interface and next-hop address and require the next hop to be directly attached.

A fully specified static route is a static route configuration that

- specifies the output interface
- explicitly identifies the next-hop address, and
- requires the next hop to be directly attached to the specified output interface.

Route validity

A fully specified route is valid when the specified interface, IPv4 or IPv6, is enabled and up.

Configure a fully specified static route

Teaches you how to configure a static route by specifying the destination prefix, output interface, and next-hop address.

1. In static route configuration mode, specify the address family.

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)#
```

2. Configure the fully specified static route with the destination prefix, outgoing interface, and next-hop address.

```
Router(config-static-afi)# 2001:0DB8::/32 FourHundredGigE 0/0/0/0
2001:0DB8:3000::1
Router(config-static-afi)#commit
```

Recursive static routes

Explains how recursive static routes resolve the output interface through a next-hop address and are withdrawn if self-recursion occurs.

A recursive static route is a static route that

- specifies only the next-hop address
- derives the output interface by resolving that next hop, and
- is withdrawn from the routing table if the system detects self-recursion.

Recursive static route behavior and self-recursion

In a recursive static route, only the next hop is specified. The output interface is derived from the next hop. A recursive static route is valid only when the specified next hop resolves, either directly or indirectly, to a valid output interface, provided the route does not self-recurse and the recursion depth does not exceed the maximum IPv6 forwarding recursion depth.

- A route self-recurses if it is itself used to resolve its own next hop.
- If a static route becomes self-recursive, RIB sends a notification to static routes to withdraw the recursive route.
- It is not normally useful to manually configure a self-recursive static route, although it is not prohibited.
- A recursive static route that has been inserted in the routing table may become self-recursive as a result of some transient change in the network learned through a dynamic routing protocol.

- If this occurs, the static route is removed from the routing table, although not from the configuration. A subsequent network change may cause the static route to no longer be self-recursive, in which case it is re-inserted in the routing table.

Recursive static route example

This example shows how to specify that all destinations with address prefix 2001:0DB8::/32 are reachable through the host with address 2001:0DB8:3000::1:

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)# 2001:0DB8::/32 2001:0DB8:3000::1
```

Self-recursive static route example

A static route is self-recursive when the next hop depends on another route, which ultimately points to the original static route.

For example, assume:

- A BGP route: `2001:DB8:3000::/48` with next hop `2001:DB8::104`
- A static route:

  `2001:DB8::/32 → 2001:DB8:3000::1`

If the next hop of the static route (2001:DB8:3000::1) is reached through the BGP route, and the BGP next hop (2001:DB8::104) is reached through the static route, a recursive loop occurs. As a result, the IPv6 RIB does not install the static route.

Floating static routes

Describes how floating static routes use a higher administrative distance so they act as backups when dynamic routes are unavailable.

A floating static route is a static route that

- uses a higher administrative distance than the dynamic route
- acts as a backup for dynamic routes, and
- is used only when the dynamic route is unavailable.

Administrative distance and route preference

By default, the router uses static routes over dynamic routes because static routes have smaller administrative distances.

Dynamic ECMP

Describes how dynamic ECMP selects from equal-cost IGP paths and supports hardware load balancing for nonrecursive prefixes.

Dynamic ECMP is a routing feature that

- dynamically selects from `1` to `64` equal-cost paths for IGP prefixes
- supports dynamic ECMP for nonrecursive prefixes, and
- enables hardware load balancing across egress links.

IPv4 multicast static routes

Explains how multicast static routes provide multicast-specific forwarding paths that differ from unicast routing and remain local to the router.

IPv4 multicast static routes are multicast routing entries that

- let multicast packets use paths that differ from unicast paths
- support networks where multicast and unicast topologies do not match, and
- allow static multicast source configuration independent of the unicast routing table.

Use multicast static routes when multicast and unicast traffic require different network paths, such as when you need to create a tunnel because some network segments do not support multicast.

- Multicast static routes remain local on your router; they are not advertised or redistributed to other routers.
- You can use multicast static routes with GRE tunnels to bypass network segments that do not support multicast routing.

Key attributes of multicast static routes

Key attributes of multicast static routes include their local scope, use in RPF checks, and ability to separate multicast from unicast routing decisions.

- Local to the router
- Used for RPF checks
- Not advertised or redistributed

Table 37: Comparison of multicast and unicast routing paths

Attributes	Multicast static route	Unicast route
Path selection	Configured by user, can use tunnels	Determined by routing table
Scope	Local to router	Network-wide

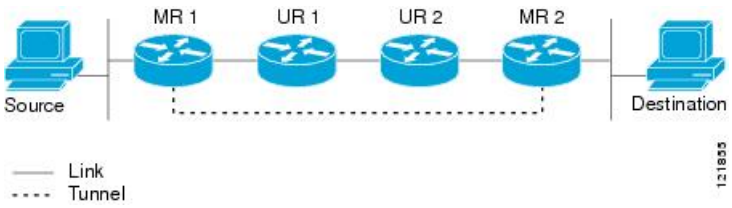
Table 38: Router Types and Supported Traffic

Router type	Unicast support	Multicast support
Unicast router	Yes	No
Multicast router	No	Yes

> **Note**
>
> Multicast static routes are not propagated to other routers and must be configured on each device as needed.

This figure illustrates the use of a GRE tunnel for multicast packets when the unicast path does not support multicast routing.

Figure 3: Tunnel for multicast packets



Multicast static routes with GRE tunnels

In your network, if there are two routers that only handle unicast traffic and two routers that support multicast, you can configure a GRE tunnel between the multicast routers so multicast packets traverse the tunnel. Unicast packets continue using the standard unicast path. With this setup, multicast traffic reaches its destination even when the unicast path does not support multicast routing.

Configure multicast static routes

Teaches you how to configure unicast and multicast static routes, enable multicast address families on interfaces, and verify multicast route entries.

1. Enable the static routing process.

```
Router# configure
Router(config)# router static
Router(config-static)#
```

2. Configure the IPv4 address-family for the unicast topology with destination prefixes.

```
Router(config-static)# address family ipv4 unicast
Router(config-static-afi)# 10.1.1.0/24 198.51.100.1
Router(config-static-afi)# 223.255.254.254/32 203.0.113.1
Router(config-static-afi)# exit
```

3. Configure the IPv4 address-family for the multicast topology with destination prefixes.

```
Router(config-static)# address-family ipv4 multicast
Router(config-static-afi)# 198.51.100.20/32 209.165.201.0
Router(config-static-afi)# 192.0.2.10/32 209.165.201.0
Router(config-static-afi)# exit
```

4. Enable IPv4 multicast and IPv6 multicast address families on the next-hop interface.

```
Router(config)# interface FourHundredGigE 0/0/0/0
Router(config-if)# address-family ipv4 multicast
Router(config-if)# address-family ipv6 multicast
Router(config-if)# commit
```

5. Verify the IPv4 multicast routes.

```
show route ipv4 multicast
Codes: C - connected, S - static, R - RIP, B - BGP, (>) - Diversion path
O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
 i - ISIS, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, su - IS-IS summary null, * - candidate default
```

```
 U - per-user static route, o - ODR, L - local, G  - DAGR, l - LISP
 A - access/subscriber, a - Application route
 M - mobile route, r - RPL, t - Traffic Engineering, (!) - FRR Backup path
Gateway of last resort is 10.1.1.20 to network 0.0.0.0
i*L1 0.0.0.0/0 [115/10] via 10.1.1.20, 00:41:12, FourHundredGigE0/0/0/6
C    10.1.1.0/24 is directly connected, 00:41:12, FourHundredGigE0/0/0/0
L    10.1.1.10/32 is directly connected, 00:41:12, FourHundredGigE0/0/0/0
S    172.16.2.10/32 [1/0] via 198.51.100.20, 00:41:12
i L1 172.16.3.1/32 [115/20] via 198.51.100.20, 00:41:12,
FourHundredGigE0/0/0/12
i L1 192.0.2.1/24 [115/20] via 198.51.100.20, 00:41:12, FourHundredGigE0/0/0/1
```

Cross address-family static routing

Describes how cross address-family static routing uses IPv6 next hops for IPv4 routes so IPv4 traffic can traverse IPv6 infrastructure.

Cross address-family static routing is a routing feature that

- supports IPv4 static routes with IPv6 next hops
- allows IPv4 traffic to travel over an IPv6 network, and
- extends static routing across address families.

Table 39: Feature History Table

Feature Name	Release Information	Feature Description
Cross address-family static routing	Release 26.1.1	With the introduction of this feature, you can now configure IPv4 prefixes with IPv6 next hops. The router programs these routes into the routing table once the Routing Information Base (RIB) confirms the next hop is reachable. This feature allows IPv4 traffic to traverse IPv6-capable infrastructure, providing a scalable solution for network management. This feature introduces these changes: **CLI :** • **router static address-family** • **show static vrf** • **show static interfaces** **YANG Data Model :** `Cisco-IOS-XR-um-router-static-cfg` (see GitHub , YANG Data Models Navigator)

Restrictions for cross address-family static routes

Provides restrictions for cross address-family routing, including unsupported IPv6-over-IPv4 use, BFD limitations, and the requirement for an IPv6 next hop.

- You can configure IPv4 static routes using an IPv6 next hop. IPv6 over IPv4 is not supported. If you attempt to configure an IPv4 next hop for an IPv6 route, you receive an 'Invalid Address Family' error.
- You cannot use Bidirectional Forwarding Detection (BFD) for cross address-family next hops.
- You cannot configure a **v4-over-v6** route using only an interface; an IPv6 next hop must be specified.

Configuration guidelines for cross address-family static routes

Explains configuration guidelines for using VRFs, recursive next hops, and interface-based next-hop paths with cross address-family static routes.

- You can configure routes within a VRF. If your destination is in a different routing table, the **show route** command lists the associated VRF.
- You can configure recursive next-hop routes or paths that include both an interface and a next hop.

Configure cross address-family static routes

Teaches you how to configure an IPv4 static route with an IPv6 next hop by using either recursive resolution or an interface-based next hop.

Follow these steps to configure cross address-family static routes. Use this task to configure an IPv4 prefix with an IPv6 next hop.

1. Enter the static router configuration mode and specify the IPv4 address family.

```
Router#configure
Router(config)# router static
Router(config-static)# address-family ipv4 unicast
Router(config-static-afi)#
```

2. Configure the Cross-Address Family Identifier (AFI) static route. You can configure this using either a recursive next-hop or a next-hop through a specific interface.
   - Recursive IPv6 next hop

     Use this option when the IPv6 next hop is reachable through another route in the routing table.

```
Router(config-static-afi)# 1.1.0.0/16 1:1:1::1
Router(config-static-afi)# commit
```

   - IPv6 next hop through an interface

     Use this option to specify exactly which interface must be used to reach the IPv6 next hop.

```
Router(config-static-afi)# 1.1.0.0/16 GigabitEthernet0/0/0/0 1:1:1::1
Router(config-static-afi)# commit
```

Verify the cross address-family static routes configuration

Teaches you how to verify route installation, next-hop resolution, and interface tracking for cross address-family static routes.

Use this task to confirm the route is correctly programmed and the next hop is resolved.

1. Check the routing table (RIB) and verify that the IPv4 prefix is present and shows the IPv6 next hop.

```
Router# show route ipv4 unicast 1.1.0.0/16
 S 1.1.0.0/16 [1/0] via 1:1:1::1 ...
```

2. Verify if the static routing process has successfully resolved the IPv6 next hop. Look for the status **Resolved** next to the IPv6 address.

```
Router# show static vrf default ipv4 next-hops
```

3. Check interface tracking (if Option B is used). Ensure the interface state is **Up** and tracked for cross-afi traffic.

```
Router# show static interfaces
```

Configuration examples

Introduces example configurations that show how to deploy static routing in common scenarios.

This section provides additional configuration examples for specific scenarios.

Configure a static route

Teaches you how to configure a static route with a next-hop interface, next-hop address, VRF context, and administrative distance.

You can configure static routes to point to a next hop interface, next hop IP address, or both. If you specify an interface, the static route is installed in the Routing Information Base (RIB) when the interface is reachable. If you do not specify an interface, the route is installed when the next hop address is reachable. When you configure a static route with the permanent attribute, it is installed in RIB regardless of reachability.

> **Tip**
>
> You can programmatically configure and view the operational state of the static route using `openconfig-local-routing.yang` or `openconfig-network-instance.yang` OpenConfig data models. To get started with using data models, see the *Programmability Configuration Guide for Cisco 8000 Series Routers* .

Use this task to configure a static route.

1. In static route configuration mode, configure the VRF.

```
Router# configure
Router(config)# router static
Router(config-static)# vrf vrf_A
Router(config-static-vrf)#
```

   If a VRF is not specified, the static route is configured under the default VRF.

2. In address family mode, configure an administrative distance of 110.

A default static route is often used in simple router topologies. This example shows how to route packets for network 10.0.0.0 through to a next hop at 192.0.2.1 if dynamic information with administrative distance less than 110 is not available.

```
Router(config-static-vrf)# address family ipv4 unicast
Router(config-static-vrf-afi)# 10.0.0.0/8 192.0.2.1 110
Router(config-static-vrf-afi)# commit
```

Associate a VRF with a static route

Teaches you how to place a static route in a specific VRF and bind the related interface to that VRF.

Use this task to associate a VRF with a static route.

1. In static route configuration mode, configure the VRF.

```
Router# configure
Router(config)# router static
Router(config-static)# vrf vrf_A
Router(config-static-vrf)#
```

2. In address family mode, configure an administrative distance of 201.

```
Router(config-static-vrf)# address family ipv6 unicast
Router(config-static-vrf-afi)# 2001:0DB8::/32 2001:0DB8:3000::1 201
```

3. Bind the interface to the VRF.

   Ensure that the interface name exactly matches the VRF name configured earlier.

```
Router(config)# interface FourHundredGigE 0/5/0/0
```

Configure a static route between PE-CE routers

Teaches you how to configure a VRF-based static route between PE and CE routers for routed connectivity.

Use this task to configure static routing between PE and CE routers.

> **Note**
>
> The system does not support VRF fallback with IPv6 VPN Provider Edge (6VPE).

1. In static route configuration mode, configure the VRF.

```
Router# configure
Router(config)# router static
Router(config-static)# vrf vrf_A
Router(config-static-vrf)#
```

2. In address family mode, configure an administrative distance of 201.

```
Router(config-static-vrf)# address family ipv4 unicast
Router(config-static-vrf-afi)# 0.0.0.0/0 192.0.2.1 120
Router(config-static-vrf-afi)# end
```

In this example, a static route between PE and CE routers is configured, and a VRF is associated with the static route:

```
configure
 router static
  vrf vrf_A
  address-family ipv4 unicast
  0.0.0.0/0 192.0.2.1 120
  end
```

Change maximum number of allowable static routes

Explains how to adjust the maximum number of static routes allowed per address family and describes how the system handles limits and excess routes.

Note

By default, you can configure up to 4000 static routes on a router for each address family. You can raise or lower the limit using the **maximum path** command.

If the maximum number of static routes for an address family is reduced below the currently configured number, the system rejects the change. If you commit a batch of routes that would exceed the maximum allowed, the first *n* routes are accepted, and the remainder are rejected. The value of *n* is the difference between the maximum number allowed and the number previously configured.

Use this task to change the maximum number of allowable static routes.

Run the **maximum path** command in static route configuration mode to change the maximum number of static routes allowed.

- Specify IPv4 or IPv6 address prefixes.
- Specify the maximum number of static routes for the given address family. The range is from 1 to 140,000.
- This example sets the maximum number of static IPv4 routes to 10,000.

```
Router(config)# router static
Router(config-static)# maximum path ipv4 10000
```

You can configure a static route to use the null 0 interface to discard traffic for a specific prefix. To discard all traffic to the prefix 2001:0DB8:42:1/64, define this static route:

```
configure
 router static
  address-family ipv6 unicast
  2001:0DB8:42:1::/64 null 0
  end
```

Configure native UCMP for static routing

Teaches you how to configure native UCMP metrics for static routes so traffic is distributed proportionally across links based on bandwidth.

Native UCMP enables effective load balancing by assigning lower metrics to higher bandwidth links, ensuring traffic is distributed according to link capacity. This approach supports both IPv4 and IPv6 static routes in virtual routing and forwarding (VRF) contexts and is preferred for multi-hop destinations.

Native UCMP:

- ensures higher-bandwidth links are utilized efficiently
- supports both local and native UCMP configurations, and
- applies to static routing scenarios requiring proportional traffic distribution.

In networks with multiple links, configuring equal metrics creates ECMP next hops, which may underutilize higher bandwidth links. Native UCMP addresses this by considering bandwidth during load balancing, especially for multi-hop destinations.

- Local UCMP uses equal metrics and does not consider bandwidth for multi-hop links.
- Native UCMP matches link metrics with end-to-end bandwidth for optimal load balancing.

Ensure you have access to the device CLI and appropriate privileges to configure static routing. Identify the interfaces and bandwidths for all links involved in the load balancing configuration.

1. Configure UCMP with load metric for IPv4 or IPv6 address family in static routing mode.

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv4 unicast
Router(config-static-afi)# 10.10.10.1/32 FourHundredGigE 0/5/0/0 metric 10
```

   For IPv6 address family, use the following sample configuration:

```
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)# 10:10::1/64 FourHundredGigE0/5/0/0 metric 10
```

2. Exit the static configuration mode and commit your configuration.

```
Router(config-static-afi)# exit
Router(config-static)# exit
Router(config)# commit
ipv4_static[1044]: %ROUTING-IP_STATIC-4-CONFIG_NEXTHOP_ETHER_INTERFACE :
Route for 10.10.10.1 is configured via ethernet interface
```

   Repeat this procedure on each router that you want to configure with native UCMP.

Native UCMP is configured for static routing, enabling proportional load balancing across links based on bandwidth. Higher bandwidth links will carry more traffic, optimizing network utilization.

Any static route does not have a default route installed. This example provides the static route configuration without the default metric specified.

```
router static
address-family ipv4 unicast
198.51.100.0/24 198.51.100.1
203.0.113.0/24 192.0.2.2
203.0.113.0/24 192.0.2.6 metric 100
```

Specify the metric value explicitly to achieve UCMP load balancing. This static route configuration example includes the default metric value.

```
router static
address-family ipv4 unicast
198.51.100.0/24 198.51.100.1
203.0.113.0/24 192.0.2.2 metric 1
203.0.113.0/24 192.0.2.6 metric 100
```

- Verify the routing table to confirm that the correct metrics are used for static routes.
- Monitor traffic distribution to ensure load balancing is functioning as expected.

8 Implementing UCMP

Topics:

Explains how unequal-cost multipath load balancing distributes traffic proportionally across paths with different costs, and describes UCMP concepts, weighted forwarding behavior, BGP and IS-IS configuration methods, scaling enhancements, and verification of normalized load-sharing in the forwarding table.

Unequal cost multipath load balancing is a forwarding capability that

- load balances traffic proportionally across multiple paths with different costs
- installs multiple paths to the same destination in the forwarding information base with an associated load metric or weight, and
- uses the load metric or weight to determine how much traffic each higher-bandwidth or lower-bandwidth path carries.

Table 40: Feature History Table

Feature Name	Release Information	Feature Description
Implementing UCMP	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.

Feature Name	Release Information	Feature Description
Implementing UCMP	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) UCMP enhances routing efficiency by allowing traffic distribution across multiple paths with different cost metrics. It effectively uses available bandwidth by distributing traffic proportionally based on path cost, optimizing network resource utilization. UCMP supports dynamic traffic load balancing, reducing congestion and improving overall network performance. This feature seamlessly integrates with existing routing protocols, providing a flexible and robust solution for complex network environments. *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 8712-MOD-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM *Previously this feature was supported on Q200 and Q100.

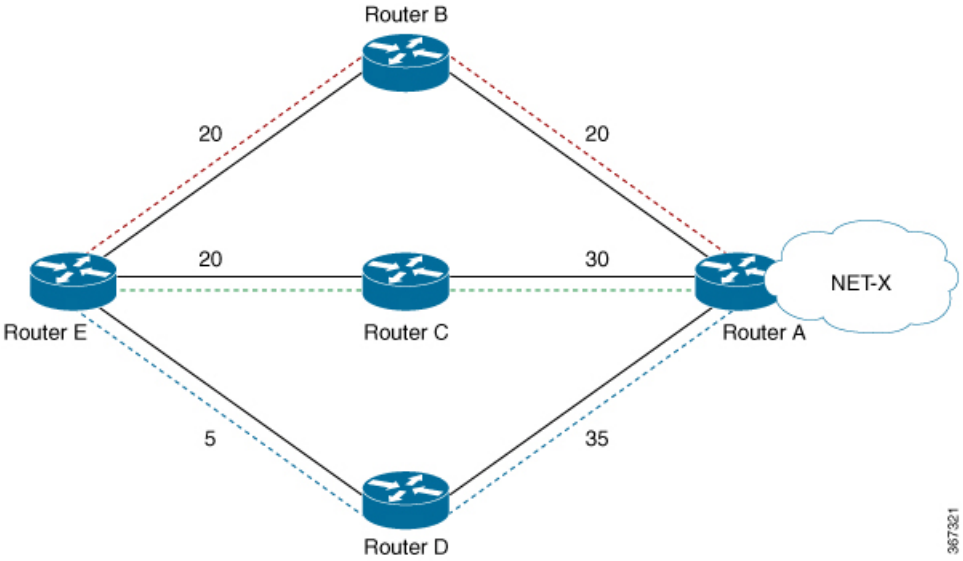
UCMP load-balancing lets you distribute traffic across multiple paths according to their cost. Generally, higher bandwidth paths have lower Interior Gateway Protocol (IGP) metrics configured, so that they form the shortest IGP paths.

When UCMP load-balancing is enabled, you can configure protocols to use lower bandwidth or higher cost paths for traffic and install them in the forwarding information base (FIB). You can install multiple paths to the same destination in FIB, and each path will have a 'load metric/weight' associated with it. FIB uses the load metric/weight to determine how much traffic is sent on each path.

Sample topology

In this example, router E has three paths to network X.

Figure 4: Topology for UCMP



This table outlines the cost associated with each path from router E to network X.

Paths	Cost from router E to network X
E-B-A	40
E-C-A	50
E-D-A	40

IGP selects the lowest-cost links, E-B-A and E-D-A. The E-C-A path is not used for load balancing because its cost is higher. The cost of the E-D link (5) does not break ties; only the total cost to network X matters.

ECMP and UCMP load balancing

Describes how ECMP distributes traffic evenly across equal-cost paths and how UCMP applies path weights to better match traffic distribution to link capacity.

ECMP load balancing is a forwarding method that

- distributes traffic across multiple equal-cost paths
- assumes the available links have similar speed and share hash buckets evenly, and
- aims to provide nearly equal load sharing across those paths.

Table 41: Feature History Table

Feature Name	Release Information	Feature Description
ECMP vs. UCMP Load Balancing	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.
ECMP vs. UCMP Load Balancing	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) Enhanced ECMP improves routing scalability by supporting a higher number of equal-cost paths, facilitating efficient load distribution across the network. This feature allows you to maximize throughput by evenly distributing traffic across available paths, which minimizes potential bottlenecks. Enhanced ECMP is designed to work seamlessly with various routing protocols, ensuring consistent performance and reliability in large-scale network environments. *This feature is supported on: - 8212-48FH-M - 8711-32FH-M - 8712-MOD-M - 88-LC1-36EH - 88-LC1-12TH24FH-E - 88-LC1-52Y8H-EM

Equal bucket distribution in ECMP

In ECMP, all available links are typically of similar speed, so hash values are shared equally across the available paths.

For example, if two paths are available, the hash buckets are divided equally for a 50 percent load share in each path. If one path is a 10G link and the other is a 1G link, equal distribution may not be desirable. In such cases, a distribution closer to 90/10 may be preferred. Because BGP does not consider bandwidth, the 10G and 1G paths are still each selected 50% of the time.

How UCMP addresses unequal links

UCMP applies a *weight* to a path, assigning more hash buckets to the path with higher weight. The applied weight is *static* , which means it is derived from the DMZ bandwidth extended community and assigned to a peer or configured using Route Policy Language (RPL) route settings.

Traffic distribution based on link capacity

A routing protocol generally selects the best path to a destination based on a metric. This metric usually depends on circuit bandwidth. For example, if there are three paths (1G, 10G, and 100G), routing protocols often use only the highest-capacity path. In some situations, you may want to distribute traffic over the circuits based on the load each can carry. For example, you might distribute traffic 1%, 10%, and 89% among the 1G, 10G, and 100G paths.

UCMP minimum integer ratios

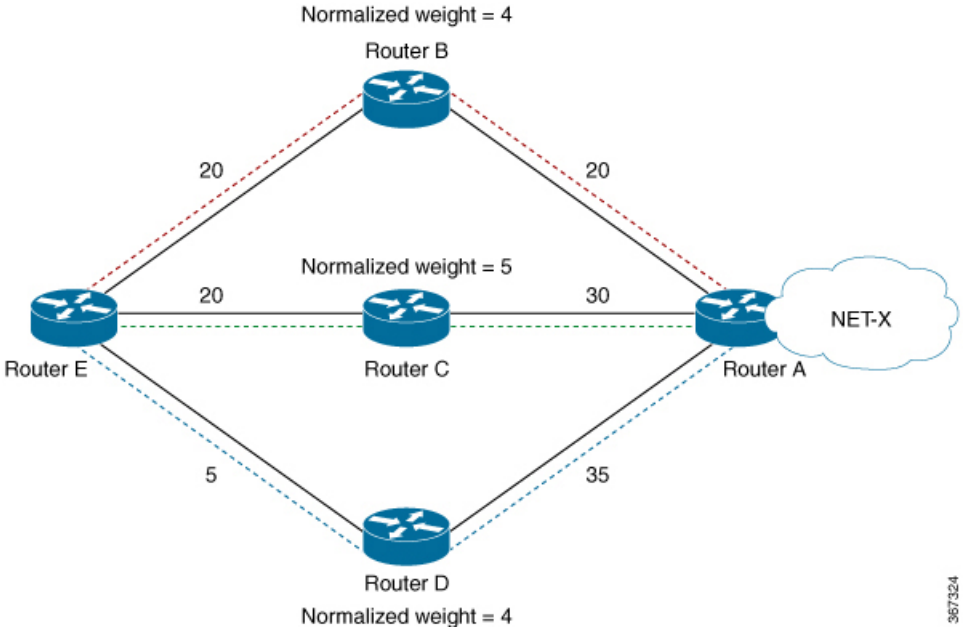
Introduces UCMP minimum integer ratios and explains the core idea and technical context. Highlights the main characteristics and usage guidance needed before configuration.

UCMP minimum integer ratios are UCMP minimum integer ratio topics that explain the UCMP Minimum Integer Ratio feature saves hardware resources when programming UCMP, by using an optimized number of buckets.

Additional details

To calculate the UCMP minimum integer ratio, find the greatest common divisor (GCD) and divide all the calculated normalized weights.

In these Figure, we have three configured weights 40, 50, and 40, with GCD as 10. To calculate the normalized weight, divide the configured weight by GCD. In this example, we need to divide 40 by 10, 50 by 10, and 40 by 10, which is 4, 5, and 4 respectively. Therefore 4, 5, and 4 are the new normalized weights.



New normalized weights are: 40/10 = 4, 50/10 = 5, and 40/10 = 4

If GCD is 1, then Normalized Weight = (Path weight/Total weight) * Maximum bucket size

BGP 256-way UCMP

Describes how BGP supports up to 256 UCMP next hops, allocates hash buckets by path weight, and improves bandwidth use and load distribution across parallel paths.

The unequal cost multipath (UCMP) load-balancing is a network traffic management method that

- balances traffic proportionally across multiple paths with different costs
- allows higher bandwidth paths to have lower Interior Gateway Protocol (IGP) metrics, and
- enables the use of lower bandwidth or higher cost paths for traffic by installing them into the forwarding information base (FIB).

The forwarding information base (FIB) is a routing component that installs multiple paths to the same destination associates each path with a load metric or weight, and uses this load metric or weight to distribute traffic across paths efficiently.

Table 42: Feature History Table

Feature Name	Release Information	Feature Description
BGP 256-way UCMP for enhanced bandwidth and load distribution	Release 25.1.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100]; Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC: Q100, Q200, P100]) You can now configure up to 256 unequal cost multipath (UCMP) next hops for BGP in both IPv4 and IPv6, significantly enhancing the previous limit of 64. This increase allows for improved network bandwidth and more effective load balancing across parallel paths.

The BGP 256-way UCMP for enhanced bandwidth and load distribution feature allows BGP to allocate 256 hash buckets across multiple paths based on assigned weights. That allocation ensures that paths with higher weights receive more hash buckets, improving bandwidth use and distributing network load more evenly.

Benefits of BGP 256-way UCMP

These are the key benefits of the BGP 256-way UCMP for enhanced bandwidth and load distribution:

- You can optimize network performance by balancing traffic across up to 256 paths. This reduces bottlenecks and improves overall data flow.
- You gain enhanced configuration flexibility and greater granularity in path configuration with up to 256 paths, enabling a wider range of weight distribution options to improve network management and optimization.

Restrictions for BGP 256-way UCMP

Provides the restriction that 256-way UCMP is not supported on MPLS networks, where only 64-way UCMP can be used.

You cannot use the BGP 256-way UCMP feature for enhanced bandwidth and load distribution on MPLS networks. If your network uses MPLS, you can only use 64-way UCMP. Consider this limitation when designing your network to ensure compatibility and optimal network performance.

Configure BGP 256-way UCMP

Teaches you how to configure IGP and BGP settings, apply bandwidth communities, and verify 256-path UCMP load distribution.

Set up the BGP 256-way UCMP for enhanced bandwidth and load distribution feature using:

- IGP configuration
- BGP configuration

This feature works with both IPv4 and IPv6 addresses. However, the examples are provided only for IPv4 addresses.

1. Configure the Interior Gateway Protocol (IGP) to enable UCMP to efficiently manage and optimize traffic flow across the network.

   Configure two interfaces for IPv4 unicast routing in point-to-point mode with weights of 127 and 129, ensuring that the combined total of paths equals 256.

```
Router(config)# router isis 1
Router(config-isis)# is-type level-2-only
Router(config-isis)# net 50.0002.0000.0000.0001.00
Router(config-isis)# address-family ipv4 unicast
Router(config-isis-af)# address-family ipv4 unicast
Router(config-isis-af)# exit
Router(config-isis)# interface FourHundredGigE0/0/0/10
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# weight 129
Router(config-isis-if-af)# exit
Router(config-isis-if)# exit
Router(config-isis)# interface FourHundredGigE0/0/0/11
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# weight 127
Router(config-isis-if-af)# exit
Router(config-isis-if)# exit
```

   You have set up two interfaces for IPv4 unicast routing in point-to-point mode with specified weights, optimizing the distribution of these weights across the 256 buckets.

2. Execute the **show cef *prefix* detail** command to verify that there are 256 multpaths available in total as configured in the IGP settings.

   The weight distribution values displayed in the show output indicates that the total number of paths is 256.

```
Router# show cef 203.0.113.254 detail
203.0.113.254/24, version 30, internal 0x1000001 0x20 (ptr 0x9c537548) [1],
0x400 (0x9c506318), 0x0 (0x0)
 local adjacency to FourHundredGigE0/0/0/10

 Prefix Len 24, traffic index 0, precedence n/a, priority 2
  gateway array (0x9c39f998) reference count 1, flags 0x0, source rib (7), 0
 backups
                [2 type 3 flags 0x8401 (0x9c444918) ext 0x0 (0x0)]
  LW-LDI[type=3, refc=1, ptr=0x9c506318, sh-ldi=0x9c444918]
  gateway array update type-time 1 Feb 18 00:23:39.241
 LDI Update time Feb 18 00:23:39.241
 LW-LDI-TS Feb 18 00:23:39.245
   via 10.0.0.2/32, FourHundredGigE0/0/0/10, 5 dependencies, weight 129, class
 0 [flags 0x0]
    path-idx 0 NHID 0x2 [0xb2f60210 0x0]
    next hop 10.0.0.2/32
    local adjacency
   via 172.31.255.254/32, FourHundredGigE0/0/0/11, 5 dependencies, weight
127, class 0 [flags 0x0]
    path-idx 1 NHID 0x1 [0xb2f60380 0x0]
    next hop 172.31.255.254/32
```

```
    local adjacency

/* These two paths carry a weight distribution designed to optimize allocation
 across 256 hash buckets. */
    slot 0, weight 129, normalized_weight 129, class 0
    slot 1, weight 127, normalized_weight 127, class 0


    Load distribution: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 (refcount
2)
```

3. Configure BGP to manage and optimize traffic distribution across multiple external paths with different costs.

   Configure route policies BW1 and BW2 to attach a bandwidth-related extended community to BGP routes, ensuring that the combined bandwidth represented by these extended communities equals a total of 256.

```
Router(config)# route-policy BW1
Router(config-route-policy)# set extcommunity bandwidth (2906:127)
Router(config-route-policy)# end-policy

Router(config)# route-policy BW2
Router(config-route-policy)# set extcommunity bandwidth (2906:129)
Router(config-route-policy)# end-policy
```

   As a result of this configuration, you set up two route policies, BW1 and BW2, to manage and optimize traffic distribution across BGP routes by adding specific bandwidth-related extended communities.

   - BW1 Policy: This policy adds an extended community with the bandwidth attribute `(2906:127)` to the BGP routes it is applied to.
   - BW2 Policy: This policy adds an extended community with the bandwidth attribute `(2906:129)` to the BGP routes it is applied to.

4. Configure the BGP router to utilize multiple paths for routing traffic, thereby enhancing load balancing and redundancy across the network.

   Set up a BGP instance with AS number 1. Adjust path selection criteria for multipath routing. Establish the IPv4 unicast address family. Enable multipath capabilities for up to 256 paths.

```
Router(config)# router bgp 1
Router(config-router)# bgp bestpath as-path multipath-relax
Router(config-router)# address-family ipv4 unicast
Router(config-router-af)# maximum-paths eibgp 256
Router(config-router-af)# exit
```

   You enable the BGP router to use 256 paths for routing traffic. This enhances load balancing and redundancy across the network.

5. Execute the **show cef *prefix* detail** command to verify that there are 256 multpaths available in total as configured in the BGP settings.

```
Router# show cef 198.51.100.254 detail
198.51.100.254/24, version 56, internal 0x5000001 0x40 (ptr 0x9c538178) [1],
 0x0 (0x0), 0x0 (0x0)
 Prefix Len 24, traffic index 0, precedence n/a, priority 4
  gateway array (0x9c3a0598) reference count 1, flags 0x2010, source rib (7),
 0 backups
                [1 type 3 flags 0x48441 (0x9c445218) ext 0x0 (0x0)]
  LW-LDI[type=0, refc=0, ptr=0x0, sh-ldi=0x0]
  gateway array update type-time 1 Feb 18 01:32:57.940
 LDI Update time Feb 18 01:32:57.940

/* These two paths carry a weight distribution designed to optimize allocation
 across 256 hash buckets. */
    Weight distribution:
    slot 0, weight 129, normalized_weight 129
    slot 1, weight 127, normalized_weight 127


  Level 1 - Load distribution: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1
```

Configure BGP with weights

Teaches you how to apply bandwidth-based route policies in BGP and verify weight-driven UCMP distribution in CEF.

Configure BGP weighted ECMP using DMZ Link Bandwidth so that the router can use multiple eBGP paths in parallel and prefer traffic sharing based on link bandwidth values.

1. Create route policies for inbound bandwidth communities and for outbound traffic pass-through.

```
Router(config)# route-policy BW1
Router(config-rpl)# set extcommunity bandwidth (2906:45750000)
Router(config-rpl)# end-policy
Router(config)# exit

Router(config)# route-policy BW2
Router(config-rpl)# set extcommunity bandwidth (2906:47250000)
Router(config-rpl)# end-policy
Router(config)# exit

Router(config)# route-policy pass-all
Router(config-rpl)# pass
Router(config-rpl)# end-policy
Router(config)# exit
```

2. Enter BGP configuration mode and enable multipath bestpath behavior.

```
Router(config)# router bgp 1
Router(config-bgp)# bgp bestpath as-path multipath-relax
Router(config-bgp)# address-family ipv4 unicast
```

```
Router(config-bgp-af)# maximum-paths eibgp 64
Router(config-bgp-af)# exit
```

3. Configure each neighbor with eBGP, enable multipath, and assign inbound and outbound route policies.

```
Router(config-bgp-af)# neighbor 1.0.0.2
Router(config-bgp-nbr)# remote-as 2
Router(config-bgp-nbr)# ebgp-multihop 255
Router(config-bgp-nbr)# dmz-link-bandwidth
Router(config-bgp-nbr)# address-family ipv4 unicast
Router(config-bgp-nbr-af)# multipath
Router(config-bgp-nbr-af)# route-policy BW1 in
Router(config-bgp-nbr-af)# route-policy pass-all out
Router(config-bgp-nbr-af)# exit
```

```
Router(config-bgp-nbr-af)# neighbor 2.0.0.2
Router(config-bgp-nbr)# remote-as 2
Router(config-bgp-nbr)# ebgp-multihop 255
Router(config-bgp-nbr)# dmz-link-bandwidth
Router(config-bgp-nbr)# address-family ipv4 unicast
Router(config-bgp-nbr-af)# multipath
Router(config-bgp-nbr-af)# route-policy BW2 in
Router(config-bgp-nbr-af)# route-policy pass-all out
```

4. Verify that the traffic is load-shared proportionally to the configured bandwidth weights.
   a) Check the CEF entry to verify correct values.

      Via 1.0.0.2: set extcommunity bandwidth (2906:45750000) – Weight = 45750000/125=366000 (125 ratio because baud)

      Via 2.0.0.2: set extcommunity bandwidth (2906:47250000) – Weight = 47250000/125=378000

      GCD is 6, so norm_weight = 61 and 63. Though 61 + 63 > 64.

   b) Check the normalized weight values to confirm calculations.

      GCD of weights 61 and 63 is 1.

      Normalised Weight = (Path weight/Total weight) * Maximum bucket size.

      The maximum bucket size value is 64. Total weight = 61+63 = 124.

      norm_weight1 = (61/124) * 64 = 31, norm_weight2 = (63/124) * 64 = 32

   c) Run the **show cef** command to verify that CEF normalized weights are derived from the configured DMZ bandwidth communities.

      Confirm CEF output reflects final normalized values.

      Confirm both next hops are installed for load sharing.

```
Router# show cef vrf default ipv4 97.0.0.0 detail
97.0.0.0/24, version 1965, internal 0x5000001 0x0 (ptr 0x71bcb620) [1],
0x0 (0x0), 0x0 (0x0)
 Updated Oct 16 08:15:02.958
 Prefix Len 24, traffic index 0, precedence n/a, priority 4
  gateway array (0x72a5e2f8) reference count 10, flags 0x2010, source rib
 (7), 0 backups
                [1 type 3 flags 0x48441 (0x71b02cd0) ext 0x0 (0x0)]
  LW-LDI[type=0, refc=0, ptr=0x0, sh-ldi=0x0]
  gateway array update type-time 1 Oct 16 08:15:02.958
 LDI Update time Oct 16 08:15:02.959
```

```
    Weight distribution:
    slot 0, weight 366000, normalized_weight 31
    slot 1, weight 378000, normalized_weight 32


  Level 1 - Load distribution: 0 1 0 1 0 1 0

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
  [0] via 1.0.0.2/32, recursive
  [1] via 2.0.0.2/32, recursive
```

The router learns multipath routes from both neighbors and tags them with different bandwidth extended communities. It can install up to 64 e/iBGP multipaths. Traffic shares proportionally–about 31:32–across the two paths, instead of equal distribution.

Configure IS-IS with weight

Teaches you how to configure IS-IS interface weights and verify normalized load distribution across multiple paths.

Configure IS-IS UCMP path weighting so two IPv4 paths are used with an explicit traffic-share ratio based on configured interface weights.

1. Configure IS-IS and set interface weights for IPv4 unicast.

```
Router(config)# router isis 1
Router(config-isis)# interface HundredGigE 0/3/0/8
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# weight 200
Router(config-isis-if-af)# exit
Router(config-isis-if)# exit
Router(config-isis)# interface HundredGigE 0/3/0/9
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# weight 300
```

2. Verify CEF entry and weighted next-hop distribution.
   a) Validate expected normalized weights from output.

```
Path weights: 200 and 300
GCD: 100
Normalized weights: 200/100 = 2, 300/100 = 3
Expected ratio: 2:3
```

   b) Confirm the values in the CEF output.

```
Router# show cef ipv4 198.51.100.3 detail
...
...
via 1.0.0.2/32 ... weight 200
via 2.0.0.2/32 ... weight 300

Weight distribution:
slot 0, weight 200, normalized_weight 2
slot 1, weight 300, normalized_weight 3
```

CEF installs both next hops and applies weighted forwarding in a 2:3 ratio (from configured weights 200 and 300).

Configure IS-IS with metric

Teaches you how to configure IS-IS metrics for UCMP and verify resulting path weights and normalized bucket distribution.

Assign different interface metrics to configure IS-IS unequal-cost multipath (UCMP). This configuration distributes traffic across multiple paths with weighted preference instead of an even split.

1. Enable IS-IS and configure IPv4 interface metric for UCMP.

```
Router# configure
Router(config)# router isis 1
Router(config-isis)# interface HundredGigE0/3/0/8
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# metric 1
Router(config-isis-if-af)# exit
Router(config-isis)# interface HundredGigE0/3/0/9
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# metric 100
```

2. Run **show cef** command to verify CEF path weights and normalized bucket distribution.
   a) Validate expected weight calculation from the output.

```
Best path metric: 21 = (configured 1 + IS-IS added 20)
UCMP path metric: 120 = (configured 100 + IS-IS added 20)

Best path weight: 0xFFFFFFFF = 4294967295
UCMP path weight: (21/120) * 4294967295 = 751619276

Normalized weights (64 buckets):
(4294967295 * 64) / (4294967295 + 751619276) = 54
(751619276 * 64) / (4294967295 + 751619276) = 9
```

   b) Confirm the values in the CEF output.

```
Router# show cef ipv4 198.51.100.0 detail
198.51.100.0/24, version 773, internal 0x1000001 0x0 (ptr 0x71bcaee0) [1],
 0x0 (0x71b98870), 0x0 (0x0)
 Updated Oct 16 06:36:08.632
 remote adjacency to HundredGigE0/3/0/8
 Prefix Len 24, traffic index 0, precedence n/a, priority 2
  gateway array (0x71a6d9d0) reference count 2, flags 0x0, source rib (7),
 0 backups
                [3 type 3 flags 0x8401 (0x71b02b90) ext 0x0 (0x0)]
  LW-LDI [type=3, refc=1, ptr=0x71b98870, sh-ldi=0x71b02b90]
  gateway array update type-time 1 Oct 16 06:36:08.632
 LDI Update time Oct 16 06:36:08.632
 LW-LDI-TS Oct 16 06:36:08.632
   via 198.51.100.2/32, HundredGigE0/3/0/8, 14 dependencies, weight
4294967295, class 0 [flags 0x0]
    path-idx 0 NHID 0x0 [0x7244d2a4 0x0]
    next hop 198.51.100.2/32
    remote adjacency
   via 198.51.100.3/32, HundredGigE0/3/0/9, 14 dependencies, weight
751619276, class 0 [flags 0x0]
    path-idx 1 NHID 0x0 [0x7244d2f8 0x0]
    next hop 198.51.100.3/32
    remote adjacency
```

```
Weight distribution:
slot 0, weight 4294967295, normalized_weight 54, class 0
slot 1, weight 751619276, normalized_weight 9, class 0
```

CEF installs both next hops and applies weighted load balancing based on computed path weights (higher share on lower-metric path). In this example (IPv4), the normalized distribution is `54:9`. The same behavior and outcome apply to IPv6 with equivalent IS-IS metric configuration.

9 Bidirectional Forwarding Detection

Topics:

Explains how BFD provides low-overhead, fast failure detection between adjacent forwarding engines across different media and protocol layers.

Bidirectional Forwarding Detection (BFD) provides low-overhead, fast detection of failures between adjacent forwarding engines. You can use BFD as a single failure-detection method across media types and protocol layers, with a wide range of detection times and overhead. Fast failure detection lets the router respond quickly to a failed link or neighbor.

Prerequisites for implementing BFD

Lists the software, protocol, platform, and neighbor support requirements needed before enabling BFD in different deployment scenarios.

Ensure you are in a user group associated with a task group that includes the proper task IDs. Consult the command reference guides for required task IDs, and contact your AAA administrator if user group assignment prevents command execution.

Implement BFD only when these system and protocol requirements are met:

- Install the required software packages: a composite PIE file including the MPLS package for MPLS, or a Cisco IOS XR IP Unicast Routing Core Bundle image for BGP, IS-IS, Static, and OSPF.
- Activate the Interior Gateway Protocol (IGP) on the router if using IS-IS or OSPF.
- Verify that the neighbor router supports BFD before enabling it for that neighbor.
- Ensure bundle member links meet specific connectivity and state requirements:
  - Connect routers on either end of the bundle back-to-back without an intervening Layer 2 switch.
  - Establish a BFD session by reaching the LACP Distributing state, configuring EtherChannel, or reaching the Hot Standby and LACP Collecting state.

Guidelines for implementing BFD

Describes ACL, port, and ICMPv6 considerations that affect BFD session establishment and operation for BGP environments.

- UMPP and LPTS ACLs apply only while a BFD session is being established and the remote discriminator is 0.
- After the session is established, BFD is offloaded to hardware, and control packets bypass UMPP ACL processing.
- Changes to UMPP ACLs after session establishment do not affect active BFD sessions.
- Use these UDP ports:
  - `3784` for single-hop BFD
  - `4784` for multihop BFD
  - `6784` for IETF BFD over bundle
- For IPv6 control-plane reachability, permit ICMPv6 if the ACL contains an implicit deny rule.

Restrictions for implementing BFD

Provides platform, feature, timer, and interoperability limitations that apply to BFD sessions, including echo mode, bundle behavior, and profile constraints.

Demand mode is not supported in Cisco IOS XR software.

Echo mode and echo-related features are restricted on specific interface types.

- Echo latency detection and echo validation are not supported on bundle interfaces.
- Echo mode is not supported on BFD over bundle interfaces and on bundle VLANS, which is BFD over logical bundle (BLB).

- BFD over unnumbered interfaces is not supported on Cisco 8000 Series routers, regardless of whether echo mode is enabled or disabled.
- Enabling Unicast Reverse Path Forwarding (uRPF) on a physical main and sub interface can cause BFD session flaps because BFD Echo mode (enabled by default) cannot coexist with uRPF on the same interface.

Resource and scaling limitations apply to BFD session configurations.

- When BFD echo mode is enabled for IPv4, the maximum supported number of BFDv4 single-hop sessions is reduced by 50%. This limitation does not apply to BFDv6 or BFDv4 multi-hop sessions.
- On centralized platforms, when the maximum number of timer profiles (8) is reached, RPFO BFD offload sessions cannot be initiated.
- The maximum Tx interval supported is 1 second. Any higher configured value is automatically adjusted to 1 second.
- A maximum of eight unique interval profiles (Desired minimum Tx interval, Required minimum Rx interval, Detection Time Multiplier) are supported across all BFD sessions.

Protocol and feature compatibility constraints must be observed.

- Virtual Router Redundancy Protocol (VRRP) and Hot Standby Router Protocol (HSRP) are not supported as BFD clients.
- If a UMPP ACL denies with logging enabled, packets are not dropped and the BFD session can establish; if it denies without logging, packets are dropped, preventing BFD session establishment.
- If peer routers have mismatched BFD configurations, with one router configured with **bundle coexistence bob-blb logical** command and the other not, the BFD session may behave unexpectedly.

The **bundle coexistence bob-blb logical** command is specific to Cisco IOS XR. Configure this command identically on all peer nodes for consistent BFD behavior.

Information about BFD

Explains core BFD operating behavior, including packet timing, failure detection, priority handling, bundle operation, dampening, hardware offload, and multipath support.

This topic introduces BFD and summarizes how it works in the network.

BFD packet intervals on physical interfaces

Describes how BFD uses echo mode and control packet timing on physical interfaces based on the configured interval.

This topic explains the BFD control packet transmission behavior on physical interfaces when echo mode is enabled.

BFD packet transmission behavior

When BFD is running over physical interfaces, echo mode is used only if the configured interval is less than two seconds.

When BFD is running over physical interfaces, the system manages packet intervals to prevent redundant high-frequency control packet processing.

These conditions apply to BFD sessions on physical interfaces:

- Fast rate control packet duplication is unnecessary because echo packets provide the required fast-rate failure detection.
- Failure detection is triggered when echo packets are not received within the echo failure detection time.

Control packet failure detection in asynchronous mode

Explains how asynchronous BFD detects failures by using negotiated intervals and multipliers to declare a neighbor down when packets stop arriving.

Control packet failure detection in asynchronous mode is a detection process that

- identifies neighbor connectivity loss by monitoring the receipt of valid control packets within a calculated time interval
- uses the minimum interval as the configured time between control packets, and
- uses the multiplier to determine the failure detection time.
- Minimum interval: The configured time duration between control packets, defined by **bfd minimum-interval** or **bfd address-family ipv4 minimum-interval** .
- Multiplier: The factor used to determine the failure detection time, defined by **bfd multiplier** or **bfd address-family ipv4 multiplier**.

Failure detection logic

The system calculates the failure detection timer based on the formula (*I* x *M*), where *I* is the negotiated interval and *M* is the multiplier provided by the remote neighbor. The local multiplier value is communicated to the neighbor to facilitate this calculation.

The detection process uses these steps:

1. The system starts a failure detection timer based on the product of the negotiated interval and the remote multiplier.
2. The system resets the timer whenever a valid control packet is received from the neighbor.
3. The system declares the neighbor down if no valid control packet is received before the timer expires.

Priority settings for BFD packets

Describes how packet priority and CoS settings help protect BFD echo packets from congestion on oversubscribed interfaces and intermediate switches.

Priority settings for BFD packets are QoS-related settings that

- assign internal priority to remote BFD Echo packets
- configure explicit CoS values in egress QoS service policies, and
- protect BFD Echo packet replies from other traffic in intermediate switches.

BFD packet priority configuration requirements

Because CoS values in ethernet headers may not be retained in Echo messages, you must explicitly configure these values to maintain traffic prioritization.

Use **set cos** command to manage CoS values for BFD packets attached to a traffic class.

For more information on configuring class-based unconditional packet marking, see *Modular QoS Configuration Guide for Cisco 8000 Series Routers*.

BFD dampening

Explains how BFD dampening suppresses repeated notifications from flapping sessions by applying configurable delays until the session becomes stable.

BFD dampening is a configurable exponential delay mechanism that

- suppresses BFD notifications until the monitored session stabilizes

- reduces processing resource consumption on network devices, and
- improves convergence time and overall network stability.

BFD dampening prevents excessive notification to BFD clients when a remote node reachability event flaps.

Table 43: Feature History Table

Feature Name	Release	Description
BFD dampening	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*)
BFD dampening	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])
BFD dampening	Release 24.4.1	Introduced in this release on: Fixed Systems(8700) (select variants only*).
BFD dampening	Release 24.2.11	Introduced in this release on: Modular Systems (8800 [LC ASIC: P100]) (select variants only*).

Bidirectional Forwarding Detection (BFD) helps routing protocols detect reachability failures quickly and notify neighboring devices immediately. When BFD detects a reachability change for a client, it notifies the client neighbors at once.

In some cases, you might need to minimize routing table changes during a microfailure to avoid affecting convergence. An unstable link that flaps repeatedly can cause other devices in the network to use significant processing resources. This condition can also cause routing protocols to lose synchronization with the state of the flapping link.

BFD dampening uses a configurable exponential delay mechanism to reduce the effects of repeated remote-node reachability flaps. It lets the network operator automatically dampen a BFD session and reduce excessive notifications to BFD clients. This behavior helps prevent unnecessary network instability. Dampening suppresses BFD notifications until the monitored session stops flapping and becomes stable.

BFD session dampening configuration

Configure BFD dampening, especially on a high-speed interface with routing clients, to improve convergence and network stability. BFD dampening applies to all BFD session types, including IPv4 single-hop, Multiprotocol Label Switching Transport Profile (MPLS-TP), and pseudowire (PW) Virtual Circuit Connection Verification (VCCV).

Configure BFD dampening at the single-hop BFD template level. Dampening applies to every session that uses that template. If you do not want dampening on a session, use a different template without dampening. By default, templates do not enable dampening.

Configure BFD dampening

Configures BFD dampening parameters to reduce session flapping. This process helps stabilize BFD sessions by introducing delays in session startup.

The dampening values can be defined for bundle member interfaces and for the non-bundle interfaces.

1. Configure the initial and maximum delay for BFD session startup on BFD bundle members.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# dampening bundle-member initial-wait 8000
Router(config-bfd)# dampening bundle-member maximum-wait 15000
```

2. Change the default initial-wait and maximum-wait for BFD on a non-bundle interface.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# dampening initial-wait 30000
Router(config-bfd)# dampening maximum-wait 35000
```

Specifies delays in milliseconds for BFD session startup to control flapping. The value for **maximum-wait** should be greater than the value for **initial-wait**.

BFD multipath sessions

Describes BFD multipath sessions on virtual interfaces and multihop paths. Use this feature to monitor connectivity across multiple paths and to support session migration during line card events.

BFD multipath sessions are BFD sessions that

- run over virtual interfaces such as GRE tunnel interfaces, PWHE interfaces, or interfaces that are multiple hops away
- support connectivity monitoring across multiple underlying paths, and
- can migrate when a hosting line card is removed or enters maintenance mode.

BFD multipath session characteristics

These events can affect BFD multipath session stability, depending on the negotiated interval and forwarding-plane convergence time:

- Failure of a path
- Online insertion or removal (OIR) of a line card which hosts one or more paths
- Removal of a link (by configuration) which constitutes a path
- Shutdown of a link which constitutes a path

Table 44: Feature History Table

Feature Name	Release	Description
BFD multipath sessions	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is supported on Cisco 8711-48Z-M routers.
BFD multipath sessions	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: 8712-MOD-M, 8011-4G24Y4H-I

Feature Name	Release	Description
BFD multipath sessions	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: 8212-48FH-M, 8711-32FH-M, 88-LC1-36EH, 88-LC1-12TH24FH-E, 88-LC1-52Y8H-EM

Configuration guidelines for BFD multipath sessions

- Configure the **bfd mutlipath include location** *location-id* command on at least one line card. This line card sends and receives packets for BFD multipath sessions.
- If you remove a line card that hosts a BFD multipath session, BFD tries to move the session to another line card. During the move, static routes are removed from the RIB. BFD then reestablishes the session and adds it back to the RIB.
- If BFD packets need priority handling, configure a QoS policy across the entire path, including ingress and egress interfaces. Classify BFD packets into priority level 1 or priority level 2 queues.
- For centralized platforms, use only MPA locations, such as 0/1, for BFD multipath configuration. Do not use line card locations, such as 0/1/CPU0, because BFD multipath does not support them.

IPv4 multihop BFD

Provides sub-second BFD failure detection for destinations up to 255 hops away. Use this feature to establish BFD sessions between nodes that are separated by multiple hops.

IPv4 multihop BFD is a BFD session that

- is established between two addresses across two nodes over paths that span multiple network hops
- supports subsecond failure detection for destinations up to `255` hops away, and
- integrates with external and internal BGP applications.

You can configure the **bfd multihop ttl-drop-threshold** command to drop packets from neighbors that exceed a specified hop count.

Table 45: Feature History Table

Feature Name	Release	Description
IPv4 multihop BFD	Release 25.4.1	Introduced in this release on Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100]).
IPv4 multihop BFD	Release 24.2.11	Introduced in this release on Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100]) and Modular Systems (8800 [LC ASIC: P100]).

IPv4 multihop BFD establishes a BFD session between two addresses on different nodes. You can use it for a BFD session between PE and CE loopback addresses or between routers that are several TTL hops apart.

Configure IPv4 multihop BFD

Configures the IPv4 Multihop BFD feature for distributed and centralized platforms. This procedure establishes BFD sessions across multiple hops to improve network convergence times.

1. Configure the BFD multihop parameters and BGP neighbor settings for the platform.

```
Router# configure
Router(config)# bfd
Router(config)# multihop ttl-drop-threshold 225
Router(config)# multipath include location 0/7/CPU0
Router(config)# router bgp 100
Router(config-bgp)# neighbor 209.165.200.225
Router(config-bgp-nbr)# remote-as 2000
Router(config-bgp-nbr)# update-source loopback 1
Router(config-bgp-nbr)# bfd fast-detect
Router(config-bgp-nbr)# bfd multiplier 3
Router(config-bgp-nbr)# bfd minimum-interval 1200
Router(config-bgp-nbr-af)# route-policy pass-all in
Router(config-bgp-nbr-af)# route-policy pass-all out
Router(config-bgp-nbr-af)# commit
```

2. Configure the BFD multihop parameters for centralized platforms if required.

```
Router# configure
Router(config)# bfd
Router(config)# multihop ttl-drop-threshold 225
Router(config)# multipath include location 0/7
Router(config)# router bgp 100
Router(config-bgp)# neighbor 209.165.200.225
Router(config-bgp-nbr)# remote-as 2000
Router(config-bgp-nbr)# update-source loopback 1
Router(config-bgp-nbr)# bfd fast-detect
Router(config-bgp-nbr)# bfd multiplier 3
Router(config-bgp-nbr)# bfd minimum-interval 1200
Router(config-bgp-nbr-af)# route-policy pass-all in
Router(config-bgp-nbr-af)# route-policy pass-all out
Router(config-bgp-nbr-af)# commit
```

3. Verify the configuration details and status of IPv4 multihop BFD using the **show tech-support bfdhwoff** command.

```
Router# show tech-support bfdhwoff
harddisk:
Tue Mar 20 11:20:29.214 PDT
++ Show tech start time: 2018-Mar-20.112029.PDT ++
Tue Mar 20 11:20:30 PDT 2018 Waiting for gathering to complete
.............................
Tue Mar 20 11:22:37 PDT 2018 Compressing show tech output Show tech output
available at 0/RP0/CPU0 :
/harddisk:/showtech-bfd-hwoff-platform-2018-Mar-20.112029.PDT.tgz
++ Show tech end time: 2018-Mar-20.112237.PDT ++
```

BFD hardware offload support for IPv4

Describes how IPv4 BFD hardware offload improves scale and convergence by moving session processing to hardware and reporting resource exhaustion through syslog messages.

BFD hardware offload is a performance feature that

- shifts BFD session processing from the CPU to the NPU on line cards

- increases the number of simultaneous BFD sessions the system supports, and
- accelerates network convergence after link failures.

With hardware offload, routers detect failures rapidly by handling BFD processing in hardware. The system now generates immediate, rate-limited syslog messages containing the relevant session handle and return code if NPU or SDK resources are depleted (Out of Resources) and the session state transitions immediately to ADMIN DOWN. These syslog notifications improve real-time monitoring, allowing you to diagnose connectivity issues efficiently and provide all required information for TAC support.

Syslog notifications

Restrictions of BFD hardware offload support for IPv4

The system generates rate-limited syslog messages when hardware resources are exhausted, including the session handle and result code for troubleshooting.

- %PKT_INFRA-BFD_OFFLOAD-4-OOR_SESSION_CREATE
- %PKT_INFRA-BFD_OFFLOAD-4-OOR_SESSION_UPDATE

- The router does not support BFD hardware offload for IPv4 over MPLS LDP, VRRP, BVI, or IRB interfaces.
- The router supports a maximum of 1,000 sessions for BFD session migration.

Configure BFD hardware offload over IPv4

Enables BFD hardware offload for multiple protocols and interfaces to improve network convergence performance. This procedure provides configuration examples for Bundle-Ether, static routes, IS-IS, OSPF, and BGP.

1. Configure BFD over Bundle (BOB) for hardware offload.

```
Router# config
Router(config)# interface Bundle-Ether 1
Router(config-if)# bfd mode ietf
Router(config-if)# bfd address-family ipv4 multiplier 3
Router(config-if)# bfd address-family ipv4 destination 192.0.2.1
Router(config-if)# bfd address-family ipv4 fast-detect
Router(config-if)# bfd address-family ipv4 minimum-interval 1200
Router(config-if)# ipv4 address 192.0.2.2/30
```

2. Configure BFD with a static route.

```
Router(config)# router static
Router(config-static)# address-family ipv4 unicast 192.0.2.0/24 192.0.2.2
bfd fast-detect minimum-interval 1200 multiplier 4
```

3. Configure BFD with IS-IS.

```
Router(config)# router isis 65444
Router(config-isis)# address-family ipv4 unicast
Router(config-isis)# exit
Router(config-isis)# interface HundredGige 0/3/0/1
Router(config-isis-if)# bfd minimum-interval 1200
Router(config-isis-if)# bfd multiplier 7
Router(config-isis-if)# bfd fast-detect ipv4
Router(config-isis-if)# address-family ipv4 unicast
```

4. Configure BFDv4 with OSPF.

```
Router(config)# router ospf main
Router(config-ospfv3)# area 0
Router(config-ospfv3-ar)# interface HundredGige 0/0/0/1
Router(config-ospfv3-ar-if)# bfd multiplier 7
Router(config-ospfv3-ar-if)# bfd fast-detect
Router(config-ospfv3-ar-if)# bfd minimum-interval 1200
```

5. Configure BFD over BGP.

```
Router(config)# router bgp 120
Router(config-bgp)# neighbor 192.0.2.1
Router(config-bgp-nbr)# bfd fast-detect
Router(config-bgp-nbr)# bfd multiplier 7
Router(config-bgp-nbr)# bfd minimum-interval 1200
```

6. Verify that BFD sessions are correctly offloaded to the hardware.

```
Router# show bfd ipv4 session
Interface           Dest Addr           Local det time(int*mult)       State

                                      Echo             Async   H/W       NPU

------------------- --------------- ---------------- ----------------
----------
Hu0/0/0/22.93       192.0.2.1       0s(0s*0)         12ms(4ms*3)      UP

                                                                 Yes   0/0/CPU0
```

BFD hardware offload support for IPv6

Explains how IPv6 BFD hardware offload improves failure detection scale and convergence while outlining support limitations for dampening and bundle operation.

BFD hardware offload support for IPv6 is a BFD capability that

- offloads BFD session processing to the network processing units on line cards in an IPv6 network
- improves scale, and
- reduces overall network convergence time by sending rapid failure detection packets to routing protocols for routing table recalculation.

Restrictions for BFD hardware offload support for IPv6

These restrictions apply to BFD hardware offload over IPv6:

- You cannot use this feature over MPLS LDP, VRRP, BVI, or IRB interfaces.
- You cannot use BFD dampening with BFD over IPv6.
- For BFD over Bundle with IPv6, you must statically configure the link-local address; dynamic configuration is not supported.

Configure BFD hardware offload for IPv6

Configures BFD hardware offload for various protocols including Bundle-Ether, static routes, IS-IS, OSPFv3, and BGP. This procedure provides specific command sequences for enabling BFD features on supported interfaces and routing processes.

These examples show how to configure BFD hardware offload for multiple protocols.

1. Configure BFD over Bundle (BOB) for hardware offload.

```
Router# config
Router(config)# interface Bundle-Ether 1
Router(config-if)# bfd mode ietf
Router(config-if)# bfd address-family ipv6 multiplier 3
Router (config-if)# bfd address-family ipv6 destination 2001:DB8:20::1
Router (config-if)# bfd address-family ipv6 fast-detect
Router(config-if)# bfd address-family ipv6 minimum-interval 1200
Router(config-if)# ipv6 address 2001:DB8:20::2/64
```

2. Configure BFD with a static route.

```
Router(config)# router static
Router(config-static)# address-family ipv6 unicast 2001:DB8:1011:17E4::1/128
 2001:DB8:AB11:15D2::2 bfd fast-detect minimum-interval 1200 multiplier 3
```

3. Configure BFD with IS-IS.

```
Router(config)# router isis 65444
Router(config-isis)# address-family ipv6 unicast
Router(config-isis)# exit
Router(config-isis)# interface HundredGige 0/3/0/1
Router(config-isis-if)# bfd minimum-interval 1200
Router(config-isis-if)# bfd multiplier 7
Router(config-isis-if)# bfd fast-detect ipv6
Router(config-isis-if)# address-family ipv6 unicast
```

4. Configure BFDv6 with OSPFv3.

```
Router(config)# router ospfv3 main
Router(config-ospfv3)# area 0
Router(config-ospfv3-ar)# interface HundredGige 0/0/0/1
Router(config-ospfv3-ar-if)# bfd multiplier 7
Router(config-ospfv3-ar-if)# bfd fast-detect
Router(config-ospfv3-ar-if)# bfd minimum-interval 1200
```

5. Configure BFD over BGP.

```
Router(config)# router bgp 120
Router(config-bgp)# neighbor 2001:DB8:1::1
Router(config-bgp-nbr)# bfd fast-detect
Router(config-bgp-nbr)# bfd multiplier 7
Router(config-bgp-nbr)# bfd minimum-interval 1200
```

Verify the configuration using the **show bfd ipv6 session** command:

```
Router# show bfd ipv6 session
Interface           Dest Addr
                                        Local det time(int*mult)      State
```

```
H/W                 NPU             Echo             Async
------------------- --------------- ---------------- ----------
BE7.2               fe80::28a:96ff:fed6:9cdb
Yes                 0/0/CPU0        0s(0s*0)         900ms(300ms*3)   UP

BE7.4               fe80::28a:96ff:fed6:9cdb
Yes                 0/0/CPU0        0s(0s*0)         900ms(300ms*3)   UP
```

Base configuration for BFD

Explains how to establish core BFD configuration and enable fast failure detection under routing protocols or static routes before applying protocol-specific settings.

To establish a BFD neighbor, complete at least one procedure to configure BFD under a dynamic routing protocol or with a static route.

Configure BFD over BGP

Shows how to configure BFD on a BGP neighbor by setting interval and multiplier values, enabling fast detection, and verifying the resulting BGP configuration.

1. Configure BFD for the BGP neighbor.

```
Router# configure
Router#(config)# router bgp 1200
Router#(config-bgp)# bfd multiplier 2
Router#(config-bgp)# bfd minimum-interval 1200
Router#(config-bgp)# neighbor 192.168.70.24
Router#(config-bgp-nbr)# remote-as 2
Router#(config-bgp-nbr)# bfd fast-detect
Router#(config-bgp-nbr)# commit
Router#(config-bgp-nbr)# end
```

2. Verify the configuration.

```
Router# show run router bgp
```

Configure BFD for OSPF

Shows how to enable BFD fast detection on an OSPF interface to reduce link-failure detection time and improve convergence.

This example shows how to enable BFD for OSPF on a HundredGigE interface.

1. Configure the BFD fast-detect feature on the OSPF interface.

```
Router# configure
Router#(config)# router ospf 0
Router#(config-ospf)# area 0
Router#(config-ospf-ar)# interface HundredGige 0/3/0/1
Router#(config-ospf-ar-if)# bfd fast-detect
Router#(config-ospf-ar-if)# commit
Router#(config-ospf-ar-if)# end
```

2. Verify the configuration.

```
Router# show run router ospf
router ospf 0
```

```
area 0
interface HundredGige 0/3/0/1
bfd fast-detect
```

Configure BFD on IPv4 static routes

Shows how to enable BFD on an IPv4 static route so the router can monitor the next hop and detect path failures quickly.

Configure the static route with BFD fast detection in IPv4 address family configuration mode..

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv4 unicast
Router(config-static)# 10.2.2.0/24 10.3.3.3 bfd fast-detect
Router(config-static)# commit
```

BFD over bundles

Enables IETF-mode fast failure detection for link-aggregation member links on a per-bundle basis.

BFD over bundle mode is a fast-failure-detection mechanism for link-aggregation member links that

- supports IETF standard configuration on a per-bundle basis
- operates without requiring system reloads or process restarts, and
- uses IANA-assigned MAC addresses and destination UDP port `6784` for IETF mode sessions.

Guidelines for BFD over bundles
Table 46: Feature History Table

Feature Name	Release	Description
BFD over bundles	Release 25.4.1	Introduced on Fixed Systems (8700, 8010). Supported on 8711-48Z-M, 8011-12G12X4Y-A, and 8011-12G12X4Y-D.
BFD over bundles	Release 25.1.1	Introduced on Fixed Systems (8010). Supported on 8011-4G24Y4H-I.
BFD over bundles	Release 24.4.1	Introduced on Fixed Systems (8700). Supported on 8712-MOD-M and 8711-32FH-M.
BFD over bundles	Release 24.2.11	Introduced on Modular Systems (8800). Supported on routers with 88-LC1-36EH line cards.

- The router supports BFD over bundles only in IETF mode.
- Use the **no bfd address-family ipv4 fast-detect** command to make BFD over bundles non-operational, or configure a bundle to `down` state by configuring **shutdown** under that particular bundle.
- Bring down and recreate existing BFD sessions to accept new BFD mode changes.

Configure BFD destination address on a bundle interface

Teaches you how to configure the primary IPv4 address of a connected remote bundle interface as the BFD destination for connectivity monitoring.

Use this task to configure the IPv4 destination address that BFD uses for connectivity monitoring on a bundle interface.

The destination address is the primary IPv4 address assigned to the connected remote bundle interface.

Replace the sample bundle ID and IPv4 address with values from your environment.

- Ensure that the bundle interface already exists and that you know the primary IPv4 address of the connected remote system.

Follow these steps to specify the BFD destination address on a bundle interface.

1. Enter interface configuration mode for the bundle interface.

```
Router# configure
Router(config)# interface Bundle-Ether 1
```

2. Configure the IPv4 destination address for BFD on the bundle interface.

   Use the primary IPv4 address of the connected remote system. The IPv4 address must be in dotted-decimal format.

```
Router(config-if)# bfd address-family ipv4 destination 10.20.20.1
Router(config-if)# commit
```

   BFD is configured to use the specified remote IPv4 destination address on the bundle interface.

The bundle interface now uses the configured remote IPv4 address as the BFD destination for connectivity monitoring.

Enable BFD sessions on bundle members

Teaches you how to enable IPv4 BFD sessions on bundle member links to improve fault detection on a bundle interface.

1. Enter the interface configuration mode for the specified bundle ID.

```
Router# configure
Router(config)# interface Bundle-Ether 1
```

2. Enable IPv4 BFD sessions on bundle member links.

```
Router(config-if)# bfd address-family ipv4 fast-detect
Router(config-if)# commit
```

Configure minimum thresholds for maintaining an active bundle

Determines whether a bundle remains active based on the minimum number of links or the minimum bandwidth available. The bundle manager monitors member link states to transition the bundle between UP and DOWN states.

The bundle manager uses two configurable minimum thresholds to determine whether a bundle can be brought up or remain up, or is down, based on the state of its member links:

- Minimum active number of links
- Minimum active bandwidth available

Whenever the state of a member changes, the bundle manager determines whether the number of active members or available bandwidth is less than the minimum. If so, then the bundle is placed, or remains, in DOWN state. Once the number of active links or available bandwidth reaches one of the minimum thresholds, then the bundle returns to the UP state.

1. Enter interface configuration mode for the bundle interface.

```
Router# configure
Router(config)# interface Bundle-Ether 1
```

2. Set the minimum amount of bandwidth required before a bundle can be brought up or remain up.

   The range is from 1 through a number that varies depending on the platform and the bundle type.

```
Router(config-if)# bundle minimum-active bandwidth 580000
```

3. Set the number of active links required before a bundle can be brought up or remain up.

   The range is from 1 to 32.

```
Router(config-if)# bundle minimum-active links 2
Router(config-if)# commit
```

> **Note**
>
> If you start BFD on an active bundle, the bundle waits for the BFD state of each active member. It then declares the bundle BFD state.

Configure BFD packet transmission intervals and failure detection times on a bundle interface

Teaches you how to configure IPv4 BFD minimum interval and multiplier values on a bundle to control packet transmission timing and failure detection thresholds.

Configure BFD asynchronous packet intervals and failure detection times for bundle member links on the bundle interface. Use the **bfd address-family ipv4 minimum-interval** command to set the BFD control-packet interval, and use the **bfd address-family ipv4 multiplier** command with that interval to set the failure detection time.

1. Enter interface configuration mode for the bundle interface.

```
Router# configure
Router(config)# interface Bundle-Ether 1
```

2. Configure the minimum interval, in milliseconds, for asynchronous mode control packets on IPv4 BFD sessions on bundle member links.

   The range is from 3 to 1200.

```
Router(config-if)#bfd address-family ipv4 minimum-interval 1200
```

3. Configure a number that is used as a multiplier with the minimum interval to determine BFD control packet failure detection times and transmission intervals for IPv4 BFD sessions on bundle member links.

   Although the command allows you to configure a minimum of 2, the supported minimum is 3.

```
Router(config-if)#bfd address-family ipv4 multiplier 30
Router(config-if)#commit
```

Configure BFD over bundles IETF mode support on a per bundle basis

Teaches you how to enable IETF mode for BFD on a specific bundle and activate IPv4 BFD sessions for that bundle.

1. Enter interface configuration mode for the bundle interface.

```
Router# configure
Router(config)# interface Bundle-Ether 1
```

2. Enable IETF mode for BFD over bundle for the specified bundle.

```
Router(config-if)# bfd mode ietf
```

3. Enable IPv4 BFD sessions on the specified bundle.

```
Router(config-if)# bfd address-family ipv4 fast-detect
Router(config-if)# commit
```

4. Run the **show bundle bundle-ether bundle-id** command to display the selected bundle mode.

BFD IPv6 in bundle manager domain

Describes how bundle manager coordinates BFD enablement and status changes for IPv6 bundle interfaces and their member links.

Use the bundle manager domain to enable or disable BFD on a bundle interface. The bundle manager applies the configuration change and requests that the BFD server enable or disable BFD on the specified bundle interfaces and their related member links.

The bundle manager uses two configurable minimum thresholds to determine whether a bundle can be brought up or remain up, or is down, based on the state of its member links.

- Minimum active number of links
- Minimum active bandwidth available

When a member state changes, the bundle manager checks whether the number of active members or the available bandwidth is below the minimum. If either value is below the minimum, the bundle stays in or moves to the DOWN state. When the number of active links or the available bandwidth reaches a minimum threshold, the bundle returns to the UP state.

BFD echo mode

Describes how BFD echo mode uses fast-rate echo packets on physical interfaces to improve link failure detection, while noting default behavior and session-scale limitations for IPv4 single-hop deployments.

BFD echo mode is a diagnostic mechanism that

- improves link failure detection by sending echo packets at fast rates
- sends echo packets more frequently than control packets to detect link failures, and
- is enabled by default for IPv4 on physical interfaces with a minimum interval of less than 3 seconds.

Table 47: Feature History Table

Feature Name	Release	Feature Description
BFD echo mode	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is supported on Cisco 8711-48Z-M routers.
BFD echo mode	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
BFD echo mode	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Note

If you enable BFD echo mode for IPv4, the maximum number of supported BFDv4 single-hop sessions drops by 50%. This limit does not apply to BFDv6 sessions or BFDv4 multihop sessions.

If you configured a BFD minimum interval greater than three seconds on a physical interface by using the **bfd minimum-interval** command, change the interval to less than three seconds before you enable echo mode. This requirement does not apply to bundle member links because they always support echo mode.

Override the default echo packet source address

Describes how to replace the default output-interface source address for BFD echo packets with a configured IPv4 address. This configuration allows for greater control over BFD packet routing and security policies.

You can override the default IP source address for BFD echo packets globally on the router or for a specific interface.

If you do not specify an echo packet source address, BFD uses the IP address of the output interface as the default source address for an echo packet.

You can use the **echo ipv4 source** command in BFD or interface BFD configuration mode to specify the IP address that you want to use as the echo packet source address.

Configure echo packet source address globally for BFD

Teaches you how to configure a router-wide IPv4 source address for BFD echo packets.

1. Enter BFD configuration mode.

```
Router(config)# bfd
```

2. Configure the IPv4 source address for BFD echo packets in the 32-bit IP address in dotted-decimal format (A.B.C.D).

```
Router(config-bfd)# echo ipv4 source 10.10.10.1
Router(config-bfd)# commit
```

Configure echo packet source address on an interface

Teaches you how to configure a specific IPv4 source address for BFD echo packets on an individual physical or bundle interface.

1. Enter BFD configuration mode.

```
Router# configure
Router(config)# bfd
```

2. Enter BFD interface configuration mode for a specific interface. In BFD interface configuration mode, you can specify an IPv4 address of an individual interface.

```
Router(config-bfd)# interface HundredGige 0/1/5/0
```

3. Configure an IPv4 address to be used as the source address in BFD echo packets in the 32-bit IP address in dotted-decimal format (A.B.C.D).

```
Router(config-bfd-if)# echo ipv4 source 10.10.10.1
Router(config-bfd-if)# commit
```

Disable echo mode

Explains how to disable BFD echo mode globally or per interface when the deployment environment does not support echo-mode operation.

Disable BFD echo mode for BFP on the router or on a specific interface when the environment does not support echo-mode operation.

- BFD does not support asynchronous operation in echo mode in certain environments.
- You can disable echo mode for BFD on the entire router, or for a particular interface.

Disable echo mode globally

Teaches you how to disable BFD echo mode globally so the router stops using echo packets.

This task describes how to disable echo mode for BFD on a router.

1. Enter BFD configuration mode.

```
Router# configure
Router(config)# bfd
```

2. Run the **echo disable** command to disable echo mode on the router.

```
Router(config-bfd)# echo disable
Router(config-bfd)# commit
```

Disable echo mode on an interface

Teaches you how to disable BFD echo mode on a specific interface from BFD interface configuration mode.

This task describes how to disable echo mode for BFD on an interface.

1. Enter BFD configuration mode.

```
Router# configure
Router(config)# bfd
```

2. Enter BFD interface configuration mode for a specific interface.

```
Router(config-bfd)# interface HundredGige 0/1/5/0
```

   In BFD interface configuration mode, you can disable echo mode on an individual interface.

3. Run the **echo disable** command to disable echo mode on the router.

```
Router(config-bfd-if)# echo disable
Router(config-bfd-if)# commit
```

Host multipath BFD sessions

Describes how multipath Bidirectional Forwarding Detection (BFD) sessions can be hosted on user-specified line cards per VRF, and explains the impact on operational resiliency and tenant isolation in multi-tenant networks.

A host multipath BFD session is a BFD session control mechanism that:

- enables assignment of multipath BFD monitoring sessions to specific line cards per VRF,
- enhances tenant isolation by preventing session overlap across VRFs sharing the same destination address, and
- improves operational resiliency and manageability for multi-tenant and managed service provider environments.

This feature is relevant when multiple tenants share network hardware but require independent operational domains, as is common in large managed service provider deployments.

Session hosting attributes and platform requirements

Table 48: Feature History Table

Feature Name	Release	Description
VRF-specific BFD multipath location assignment	Release 26.2.1	This feature allows you to pin BFD MP sessions to specific LCs by associating a destination IP and VRF with a physical location. When configured, the system automatically migrates matching sessions to the designated LC. This prevents collateral service disruption for unrelated tenants by ensuring that an LC reload only impacts sessions pinned to that specific hardware.

In the default configuration, multipath BFD sessions are hosted only by destination address. When multiple VRFs use the same BFD destination, sessions may overlap, reducing isolation.

The main attributes of per-VRF line card BFD session hosting include:

- Session granularity: Assigns BFD sessions per VRF per line card.
- Tenant isolation: Prevents operational dependencies among VRFs sharing hardware and BFD sessions.
- Resiliency: Limits the impact of hardware events to affected VRFs only.
- Management flexibility: Supports per-customer or per-service resource planning and scaling.
- Session support: Applies to both multi-hop and single-hop BFD multipath sessions

Comparison of BFD session hosting behavior:

Table 49: Comparison of default and per-VRF BFD session hosting

Attribute	Default hosting	Per-VRF and line card hosting
Session placement	Based on destination address only	Based on destination and VRF; explicitly assignable to line cards
Tenant/session isolation	Possible session overlap across VRFs or tenants	Strict isolation per VRF and line card
Operational resiliency	Single line card failure impacts all sessions sharing destination	Line card failure impacts sessions for assigned VRF only

> **Tip**
>
> Map different VRFs to distinct line cards to maximize tenant isolation and improve service reliability, particularly in shared infrastructure environments.

Host multipath BFD sessions platform support and restrictions

Lists the supported platforms, software requirements, and operational restrictions for host multipath BFD sessions with per-VRF line card assignment on Cisco ASR 9000 Series routers.

This reference provides the relevant operational restrictions for these features.

The following items describe prerequisites, and feature applicability for host multipath BFD sessions with per-VRF line card mapping.

- Per-VRF LC assignment is applicable to both single-hop and multi-hop BFD multipath sessions. Single-hop BFD without multipath is not affected by this feature.
- The `multipath include location` command must be configured before specifying per-VRF `multipath destination` assignments.

Best practice: Host multipath BFD sessions on configured LC per VRF

Outlines best practices for assigning and operating multipath Bidirectional Forwarding Detection (BFD) sessions per VRF, optimizing line card placement for isolation, resiliency, and manageability on Cisco ASR 9000 Series routers.
Configuration best practices

Configure `multipath include location` before adding per-VRF `multipath destination` entries. Remove all per-VRF `multipath destination` entries before unconfiguring `multipath include location` to prevent orphaned sessions and ensure mapping integrity.

- Specify the destination IP, VRF name, and LC location for each BFD multipath session to ensure accurate assignment.
- Schedule configuration changes during a maintenance window when possible. Always verify mapping using `show bfd multipath` and `show bfd session details` after updates.

Operational isolation and redundancy

Assign each VRF to separate line cards whenever possible. Distribute VRFs and sessions across multiple LCs to maximize isolation and minimize service disruption during line card events.

- Use clear mapping conventions for VRFs, destinations, and LC locations to aid documentation and troubleshooting.
- Regularly verify current multipath BFD assignments with `show bfd multipath` and `show bfd session details`, especially after configuration changes or hardware events.

Applies to routers running multipath BFD with multiple line cards and VRF-based segmentation, especially in multi-tenant and managed service environments.

Configure host multipath BFD session on a line card per VRF

Configure host multipath BFD session on line cards per VRF to control placement and isolation of BFD sessions for specific VRFs.

Configure host multipath Bidirectional Forwarding Detection (BFD) sessions on specific line cards (LCs) per VRF to provide per-tenant isolation and flexible placement for enhanced resiliency.

- Improve operational resilience by isolating BFD sessions for distinct VRFs on dedicated LCs.

- Enable VRF-specific resource planning and maintenance of BFD sessions.

This task is performed in multi-tenant or managed service provider deployments where multiple VRFs share destination addresses and need distinct session hosting. Hosting BFD multipath sessions on user-specified LCs per VRF reduces cross-tenant impact and gives greater placement control.

- Supported on Cisco ASR 9000 Series routers with VRF segmentation and multiple line cards.
- Affects both multi-hop and single-hop BFD multipath sessions.

Before you begin, ensure you have access and all resources are in place:

- You have administrative privileges on the target device.
- You have identified the destination IP address and VRF to be mapped to a line card.
- The intended LC is already included using the `multipath include location` command.

Follow these steps to configure host multipath BFD session on a line card per VRF.

1. Enter BFD configuration mode. Assign the specified BFD multipath session for the destination and VRF to the designated line card.

```
Router(config)# bfd
Router (config)# multipath include location 0/1/CPU0
Router (config)# multipath destination 10.1.1.1 vrf vrf1 location 0/1/CPU0
```

   Assigns the specified BFD multipath session for the destination and VRF to the designated line card.

   - The `location` must match an LC included by `multipath include location`.
   - If `vrf` is omitted, the session applies to the default VRF.

2. Remove a per-VRF and LC assignment.

```
Router (config)# no multipath destination destination 10.1.1.1 vrf vrf1 vrf
A location 0/1/CPU0
Router (config)# multipath include location 0/1/CPU0
```

   Multipath BFD session assignments and hosting are validated.

3. Verify multipath BFD session assignments and hosting.

```
show bfd multipath
```

```
show bfd session details
```

   Multipath BFD session assignments and hosting are validated.

Multipath BFD sessions are hosted on the assigned line cards per VRF, providing granular tenant monitoring and improved operational resilience.

Monitoring and verification of BFD

Describes how to verify BFD session health, inspect packet statistics, identify anomalies, and reset counters for ongoing monitoring and troubleshooting.

This topic provides the necessary commands and methods to verify session establishment, monitor packet transmission, and troubleshoot BFD session failures.

BFD provides counters and statistics that help you:

- Verify that sessions are established and functioning correctly
- Monitor packet transmission and reception
- Identify packet loss or anomalies
- Troubleshoot session flaps or failures

You can also reset or clear counters to establish a new baseline for monitoring and analysis.

Clear and display BFD counters

Teaches you how to view, clear, and verify BFD packet counters for sessions on specific nodes or interfaces.

You can clear packet counters for BFD sessions that are hosted on a specific node or on a specific interface.

1. Display the BFD packet counters.

```
Router# show bfd counters all packet location 0/RP0/CPU0
```

2. Clear the BFD packet counters.

```
Router# clear bfd counters all packet location 0/RP0/CPU0
```

3. Verify that the BFD packet counters are cleared.

```
Router# show bfd counters all packet location 0/RP0/CPU0
```

Feature-specific integrations for BFD

Describes how BFD integrates with specialized transport and forwarding environments such as MPLS TE, VXLAN, BVI, and PWHE, along with related platform-specific features and operational considerations.

Provides details on BFD integration with specific technologies, including configuration and verification procedures for fast failure detection.

BFD with MPLS Traffic Engineering (RSVP-TE)

Explains how BFD supports MPLS TE and RSVP deployments through hardware offload, LSP monitoring, tail-end operation, tunnel timing controls, dampening, and verification workflows.

This section describes the configuration and management of BFD for MPLS Traffic Engineering tunnels, including the adjustment of timers, dampening, and monitoring parameters.

BFD supports MPLS Traffic Engineering (TE) tunnels to enable fast failure detection for label-switched paths (LSPs). You can configure BFD for TE tunnels, including head-end and tail-end operations, and tune parameters such as timers, dampening, and monitoring.

BFD hardware offload for RSVP tail-end

Describes how hardware-offloaded BFD improves scale and convergence for RSVP tail-end MPLS LSP monitoring by detecting data-plane failures rapidly with lower CPU overhead.

BFD hardware offload for RSVP tail-end is a mechanism that

- offloads BFD control packet processing from the CPU to hardware for high-scale monitoring of MPLS LSP data plane health
- supports rapid failure detection for a large number of LSPs, and

- reduces CPU overhead compared to traditional LSP ping-based verification.

Table 50: Feature History Table

Feature Name	Release	Description
BFD hardware offload for resource reservation protocol tail-end	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.
BFD hardware offload for resource reservation protocol tail-end	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: 8212-48FH-M, 8711-32FH-M, 8712-MOD-M, 88-LC1-36EH, 88-LC1-12TH24FH-E, 88-LC1-52Y8H-EM.
BFD hardware offload for resource reservation protocol tail-end	Release 7.9.1	Initial support for BFD over Tail-End LSP feature.

Use Bidirectional Forwarding Detection (BFD) to detect Multiprotocol Label Switching (MPLS) Label Switched Path (LSP) data-plane failures. Use an LSP ping request to detect MPLS data-plane failures and to verify the data plane against the control plane. BFD does not verify the data plane against the control plane. However, BFD control packets require less control-plane processing than LSP ping messages. As a result, you can use BFD for faster detection of data-plane failures across many LSPs.

This feature improves scale and reduces overall network convergence time by sending rapid failure-detection packets to routing protocols so they can recalculate the routing table.

BFD over MPLS Traffic Engineering LSPs

Explains how BFD detects MPLS TE LSP data-plane failures with lower control-plane overhead than LSP ping, supporting fast tunnel protection, re-optimization, and large-scale monitoring.

The BFD over MPLS TE LSPs feature provides a mechanism for detecting MPLS data plane failures using BFD, which requires less control plane processing than LSP Ping messages. This implementation is based on *RFC 5884* and supports faster data plane failure detection when combined with LSP Ping.

BFD over MPLS tail-end LSPs supports these features:

- BFD asynchronous mode. (BFD echo mode is not supported.)
- IPv4 only, because the MPLS core uses IPv4.
- BFD packets with IP DSCP 6 (Internet Control).
- BFD for TE tunnel bring-up, reoptimization, and path protection, including standby and FRR.
- A fast detection time of 9 ms by using a 3 ms interval and a multiplier of 3.
- Optional periodic LSP ping verification after the BFD session comes up.
- Dampening to hold down a BFD failed path option.

The BFD process interacts with the tail-end and LSPV processes to support the BFD over TE LSP feature, where BFD packets from tail-end to head-end are handled in two ways:

- BFD packets from tail-end to head-end are IP routed .
- BFD packets from tail-end to head-end are label switched if MPLS LDP is available in the core with a label path from tail-end to head-end.

Configure BFD over MPLS TE tunnels

Teaches you how to enable BFD on MPLS TE tunnels at the head-end by configuring fast detection, interval, and multiplier settings for tunnel monitoring.

BFD for TE tunnel is enabled at the head-end by configuring BFD parameters under the tunnel. When BFD is enabled on the already up tunnel, TE waits for the bringup timeout before bringing down the tunnel.

> **Note**
>
> BFD paces the creation of BFD sessions by limiting LSP ping messages to be under 50 PPS to avoid variations in CPU usage.

1. Enter global configuration mode and configure MPLS OAM.

```
Router# config
Router(config)# mpls oam
```

2. Configure MPLS TE tunnel interface and enter MPLS TE tunnel interface configuration mode.

```
Router(config)#interface tunnel-te 65535
```

3. Enable BFD fast detection.

```
Router(config-if)#bfd fast-detect
```

4. Configure hello interval in milliseconds.

```
Router(config-if)#bfd minimum-interval 500
```

> **Note**
>
> Hello interval range is 3 to 1000 milliseconds. Default hello interval is 100 milliseconds.

5. Configure BFD multiplier detection.

```
Router(config-if)#bfd multiplier 5
Router(config-if)#commit
```

> **Note**
>
> BFD multiplier range is 3 to 10. Default BFD multiplier is 3.

Configure BFD bring up timeout

Teaches you how to configure the wait time before the system declares that a BFD session on a TE tunnel failed to come up.

1. Enter global configuration mode and configure MPLS OAM.

```
Router# config
Router(config)# mpls oam
```

2. Configure MPLS TE tunnel interface and enter MPLS TE tunnel interface configuration mode.

```
Router(config)#interface tunnel-te 65535
```

3. Configure the time interval in seconds to wait for the BFD session to come up.

```
Router(config-if)#bfd bringup-timeout 2400
Router(config-if)#commit
```

> **Note**
>
> The timeout range is 6 to 3600 seconds. Default bring up timeout interval is 60 seconds.

Configure BFD dampening for TE tunnels

Teaches you how to configure dampening delay intervals for TE tunnels so failed path-options back off and signaling churn is reduced.

When BFD session fails to come up, TE exponentially backs off using the failed path-option to avoid signaling churn in the network.

1. Enter global configuration mode and configure MPLS OAM.

```
Router# config
Router(config)# mpls oam
```

2. Configure MPLS TE tunnel interface and enter MPLS TE tunnel interface configuration mode.

```
Router(config)#interface tunnel-te 65535
```

3. Configure the initial delay interval before bringing up the tunnel.

```
Router(config-if)#bfd dampening initial-wait 360000
```

> **Note**
>
> The initial-wait bring-up delay range is 1 to 518,400,000 milliseconds. The default initial-wait interval is 16,000 milliseconds.

4. Configure the maximum delay interval before bringing up the tunnel.

```
Router(config-if)#bfd dampening maximum-wait 700000
```

> **Note**
>
> The maximum-wait bring-up delay range is 1 to 518,400,000 milliseconds. The default maximum-wait interval is 600,000 milliseconds.

5. Configure the secondary delay interval before bringing up the tunnel.

```
Router(config-if)#bfd dampening secondary-wait 30000
Router(config-if)#commit
```

> **Note**
>
> The secondary-wait bring-up delay range is 1 to 518,400,000 milliseconds. The default secondary-wait interval is 20,000 milliseconds.

Configure periodic LSP ping requests

Teaches you how to configure periodic LSP ping requests after a BFD session is established on an MPLS TE tunnel.

1. Configure MPLS TE tunnel interface and enter MPLS TE tunnel interface configuration mode.

```
Router# config
Router(config)#interface tunnel-te 65535
```

2. Set the periodic interval for LSP ping requests in seconds.

```
Router(config-if)#bfd lsp-ping interval 300
Router(config-if)#commit
```

> **Note**
>
> The interval range is 60 to 3,600 seconds. Default interval is 120 seconds.

Configure BFD at the tail-end

Teaches you how to configure global tail-end BFD interval and multiplier settings for all BFD over LSP sessions.

The ranges and default values are the same as the BFD head-end configuration values.

1. Enter global configuration mode and configure MPLS OAM.

```
Router# config
Router(config)# mpls oam
```

2. Configure hello interval in milliseconds.

```
Router(config)#mpls traffic-eng bfd lsp tail minimum-interval 500
```

> **Note**
>
> Hello interval range is 3 to 1,000 milliseconds. The default hello interval is 100 milliseconds.

3. Configure BFD multiplier detection.

```
Router(config)#mpls traffic-eng bfd lsp tail multiplier 5
Router(config-if)#commit
```

> **Note**
>
> BFD multiplier detect range is 3 to 10. The default BFD multiplier is 3.

Configure BFD over LSP sessions on line cards

Teaches you how to enable the required hosting location for BFD over LSP sessions by configuring multipath support on a valid line-card, RP, or MPA location.

BFD over LSP sessions, including head-end and tail-end sessions, are hosted on line cards with this configuration enabled.

> **Note**
>
> On fixed and centralized platforms, there are no line cards (LCs) in the data path. BFD sessions run on the routing processors (RPs). On fixed platforms, specify the RP location in the configuration. On centralized platforms, RP locations cannot be used in the configuration even though BFD runs on the RPs. Instead, you must specify a valid MPA location, such as 0/1, for BFD multipath configuration. LC locations, such as 0/1/CPU0, are not supported and must not be used.

In BFD configuration mode, configure BFD multiple path on a specific line card.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# multipath include location 0/1/CPU0
Router(config-if)#commit
```

Configure BFD over MPLS TE tunnel tail-end

Shows how to configure a minimum tail-end BFD interval for MPLS TE tunnels and verify session status at both the tail-end and head-end.

1. Enter global configuration mode.

```
Router# configure terminal
```

2. Configure the BFD minimum interval for the MPLS TE LSP tail-end.

```
Router(config)# mpls traffic-eng bfd lsp tail minimum-interval 3
Router(config)# commit
```

3. Run the **show bfd session** command to verify the configuration on the tail-end.

```
Router# show bfd session
Src Addr            Dest Addr       VRF Name         Type Specific Data

                                        Local det time(int*mult)      State

H/W                 NPU                 Echo             Async
------------------- --------------- ---------------- ----------------
----------
1.1.1.1             2.2.2.2         default          TT32768 (LSP:2)

                                    n/a              1500ms(500ms*3)   UP
```

4. Run the **show bfd label session** command to verify the configuration on the head-end.

```
Router# show bfd label session
Interface           Label               Local det time(int*mult)      State

                                    Echo             Async
H/W                 NPU
------------------- --------------- ---------------- ----------------
----------
tt1 (LSP:103)       24001           n/a              150ms(50ms*3)    UP

Yes                 0/1/CPU0
tt2 (LSP:102)       24002           n/a              150ms(50ms*3)    UP
```

```
Yes                  0/1/CPU0
tt3 (LSP:101)        24004           n/a               150ms(50ms*3)     UP

Yes                  0/1/CPU0
tt4 (LSP:103)        24005           n/a               150ms(50ms*3)     UP

Yes                  0/1/CPU0
tt5 (L      P:104)       24006           n/a               150ms(50ms*3)
UP
Yes                  0/1/CPU0
```

BFD over VXLAN deployments

Describes how BFD integrates with VXLAN to monitor tunnel health, detect failures quickly, and support BGP- or static-route-based VXLAN deployments within scale and mode limitations.

This topic provides configuration details, restrictions, and examples for deploying BFD in VXLAN environments.

Bidirectional forwarding detection over VXLAN tunnels

Describes how BFD monitors VXLAN tunnel health to detect Layer 2 data-plane failures quickly and support rapid failover in BGP- or static-route-based VXLAN deployments.

Bidirectional forwarding detection over VXLAN is a monitoring mechanism that

- detects failures in VXLAN tunnels to improve network reliability
- provides rapid detection of Layer 2 data plane failures, and
- supports integration with BGP or static routes for path management and fast rerouting.

Table 51: Feature History Table

Feature Name	Release Information	Feature Description
Bidirectional forwarding detection over VXLAN tunnel	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I
Bidirectional forwarding detection over VXLAN tunnel	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

Feature Name	Release Information	Feature Description
Bidirectional forwarding detection over VXLAN tunnel	Release 24.2.11	You can now monitor the health of VXLAN tunnel and detect failures in the tunnel rapidly which ensures faster rerouting of traffic, resulting in high availability of networks.

Note

BFD over VXLAN with BGP supports up to 2000 BFD sessions (IPv4 and IPv6 combined), while BFD over VXLAN on static routes supports up to 10,000 BFD sessions (up to 10 percent IPv6 and 90 percent IPv4).

How VXLAN network traffic management works

Describes how VXLAN encapsulates traffic across Layer 3 infrastructure, uses routing information for path selection, and relies on BFD to detect failures and trigger rerouting.

A VXLAN network encapsulates network traffic to form secure tunnels across the Layer 3 physical infrastructure. These tunnels maintain traffic privacy and security by isolating data from the internal domain network.

These are the key components involved in the process:

- VXLAN: Encapsulates traffic and performs load balancing to maintain performance.
- BGP or static routes: Facilitates the sharing of route information.
- BFD: Monitors tunnels to detect link failures and trigger rerouting.

These stages describe the operational flow of VXLAN traffic management.

1. Traffic distribution and routing configuration.
   - VXLAN load balancing distributes data packets to maintain performance as traffic volume increases.
   - Route information is shared using either BGP for dynamic learning or static routes for predefined paths.
2. Failure detection and recovery.

When...	And...	Then...	And...
A link failure occurs	BFD is integrated	BFD detects the failure	Traffic is rerouted to minimize downtime

Restrictions for BFD over VXLAN

Provides scale, mode, platform, interval, and traffic-rate limitations for BFD over VXLAN, including multihop-only support and asynchronous-only operation.

Use multihop BFD sessions to achieve the maximum supported scale. Maintain a distribution ratio of 90 percent IPv4 sessions to 10 percent IPv6 sessions to ensure the maximum supported scale is attainable.

- The maximum supported scale is 10,000 BFD sessions, supported only on Cisco 8100 and 8200 series routers.
- The maximum supported scale may not be attainable if the configuration includes other types of BFD sessions or if the proportion of IPv6 sessions exceeds 10 percent.
- Operate BFD over VXLAN exclusively in BFD asynchronous mode; echo mode is not supported.

- Avoid the parallel operation of BFD over VXLAN with static routes at a scale of up to 10,000 sessions alongside BFD over VXLAN with BGP, as this configuration is not supported.
- Configure a minimum-interval value greater than 300 milliseconds for software-based BFD sessions to maintain stability at higher scales.
- Keep the maximum rate in packets-per-second (PPS) for BFD sessions below 80% to ensure session stability.
- Exceeding the 80% PPS threshold may result in unstable sessions.
- Use the **show bfd summary** command to monitor the current PPS percentage.

Configure BFD over VXLAN

Describes the supported configuration scenarios for BFD over VXLAN, including deployments that use BGP or static routes for tunnel path monitoring.

This topic provides the configuration scenarios for implementing BFD over VXLAN.

The BFD over VXLAN feature can be configured for these scenarios:

- BFD over VXLAN with BGP
- BFD over VXLAN with static routes

Configure BFD over VXLAN with BGP

Shows how to configure BFD over VXLAN in a BGP-based deployment by setting up VRF, multipath BFD, interfaces, NVE, and BGP neighbor parameters.

The configuration of BFD over VXLAN with BGP includes these steps:

- VRF configuration
- Multipath BFD sessions configuration
- Interfaces configuration
- Interface NVE configuration
- BGP configuration
- BFD configuration

1. Configure the VRF.

```
Router# configure
Router(config)# vrf vrf1
Router(config-vrf)# exit
```

2. Configure the multipath BFD sessions.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# multipath include location 0/0/CPU0
Router(config-bfd)# exit
```

3. Configure the interfaces.

```
Router(config)# interface Loopback 0
Router(config-if)# ipv4 address 10.10.10.10 255.0.0.0
```

```
Router(config-if)# exit
Router(config)# interface Loopback 1
Router(config-if)# vrf vrf1
Router(config-if)# ipv4 address 192.168.0.0 255.255.0.0
Router(config-if)# exit
```

4. Configure interface NVE for decapsulation.

```
Router(config)# interface nve1
Router(config-if)# member vni 2
Router(config-nve-vni)# vrf vrf1
Router(config-nve-vni)# host-reachability protocol static
Router(config-nve-vni)# exit
Router(config-if)# overlay-encapsulation vxlan
outer(config-nve-encap-vxlan)# peer-ip lookup disable
Router(config-nve-encap-vxlan)# exit
Router(config-if)# source-interface Loopback1
Router(config-if)# commit
```

5. Configure BGP.

```
Router(config)# router bgp 1
Router(config-bgp)# bgp router-id 10.10.10.10
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# redistribute connected
Router(config-bgp-af)# exit
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp)# exit
Router(config)# router bgp 1
Router(config-bgp)# vrf vrf1
Router(config-bgp-vrf)# rd auto
Router(config-bgp-vrf)# address-family ipv4 unicast
Router(config-bgp-vrf-af)# redistribute connected
Router(config-bgp-vrf-af)# redistribute static
Router(config-bgp-vrf-af)# exit
```

6. Configure BFD over BGP.

```
Router(config)# router bgp 1
Router(config-bgp)# neighbor 10.6.6.1
Router(config-bgp-nbr)# bfd fast-detect
Router(config-bgp-nbr)# bfd multiplier 7
Router(config-bgp-nbr)# bfd minimum-interval 1200
Router(config-bgp-nbr)# remote-as 300
Router(config-bgp-nbr)# ebgp-multihop 255
Router(config-bgp-nbr)# update-source loopback 1
Router(config-bgp-nbr)# address-family ipv4 unicast
Router(config-bgp-nbr-af)# route-policy pass-all in
Router(config-bgp-nbr-af)# route-policy pass-all out
Router(config-bgp-nbr-af)# commit
```

Configure BFD over VXLAN with static routes

Shows how to configure BFD over VXLAN in a static-route-based deployment by setting up VRFs, multipath BFD, interfaces, NVE, VXLAN static routes, and session verification.

The configuration of BFD over VXLAN with static routes includes these steps:

- VRF configuration
- Multipath BFD sessions configuration
- Interfaces configuration
- Interface NVE configuration
- Static routing configuration

1. Configure the VRF.

```
Router# configure
Router(config)# vrf vrf1
Router(config-vrf)# exit
```

2. Configure the multipath BFD sessions.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# multipath include location 0/0/CPU0
Router(config-bfd)# exit
```

3. Configure the interfaces.

```
Router(config)# interface Loopback 0
Router(config-if)# ipv4 address 10.10.10.10 255.0.0.0
Router(config-if)# ipv6 address 2001:DB8:1::1/32
Router(config-if)# exit
Router(config)# interface Loopback 1
Router(config-if)# vrf vrf1
Router(config-if)# ipv4 address 192.168.0.0 255.255.0.0
Router(config-if)# exit
Router(config)# interface TenGigE0/0/0/0/0
Router(config-if)# ipv4 address 10.12.13.10 255.0.0.0
Router(config-if)# ipv6 address 2001:DB8:13::11/16
Router(config-if)# exit
```

4. Configure the interface NVE for decapsulation.

```
Router(config)# interface nve1
Router(config-if)# member vni 2
Router(config-nve-vni)# vrf vrf1
Router(config-nve-vni)# host-reachability protocol static
Router(config-nve-vni)# exit
Router(config-if)# overlay-encapsulation vxlan
Router(config-nve-encap-vxlan)# exit
Router(config-if)# source-interface Loopback1
Router(config-if)# commit
```

5. Configure the static routes.

```
Router# configure
Router(config)# router static
```

```
Router(config-static)# address-family ipv4 unicast
Router(config-static-afi)# 10.10.10.10/32 10.151.11.2
Router(config-static-afi)# exit
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)# 2001:DB8:1::12/32 2001:DB8:17::122
Router(config-static-afi)# exit
Router(config-static)# vrf VRF1
Router(config-static-vrf)# address-family ipv4 unicast
Router(config-static-vrf-afi)# 10.1.1.1/32 remote-next-hop 10.13.19.10 tunnel
 VXLAN index 1 nve 1 evni 1 src-mac aaa1.bbb1.ccc1 ->  IPv4 over IPv4
Router(config-static-vrf-afi)# 209.165.201.0/27 10.1.1.1 bfd fast-detect
minimum-interval 1000 multihop 192.168.0.0
Router(config-static-vrf-afi)# 10.12.12.12/32 remote-next-hop 10.13.19.10
tunnel VXLAN index 2 nve 1 evni 2 src-mac aaa1.bbb1.ccc2 ->  IPv4 over IPv4

Router(config-static-vrf-afi)# 209.165.202.129/27 10.12.12.12 bfd fast-detect
 minimum-interval 1000 multihop 192.168.12.24
Router(config-static-vrf-afi)# exit
```

6. Verify the BFD session details using these commands.

- ```
  Router# show bfd all
  Tue Dec 12 10:07:03.395 UTC

  IPv4:
  -----
  IPV4 Sessions Up: 7200, Down: 0, Unknown/Retry: 2000, Total: 9200

  IPv6:
  -----
  IPV6 Sessions Up: 800, Down: 0, Unknown/Retry: 0, Total: 800

  Label:
  ------
  Label Sessions Up: 0, Down: 0, Unknown/Retry: 0, Total: 0
  ```

- ```
  Router# show bfd ipv6 multihop session
  Tue Mar 26 19:32:14.851 UTC
  Src Addr                                  Dest Addr

  VRF Name                                Local det time(int*mult)      State

                                       Echo             Async
  H/W                  NPU
  -------------------- --------------- ---------------- ----------------
  ----------
  2001:DB8:0000::1                          2001:DB8:FFF::1

  vrf5001                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::2                          2001:DB8:FFF::2

  vrf5002                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::3                          2001:DB8:FFF::3

  vrf5003                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::4                          2001:DB8:FFF::4

  vrf5004                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::5                          2001:DB8:FFF::5

  vrf5005                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::6                          2001:DB8:FFF::6

  vrf5006                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::7                          2001:DB8:FFF::7

  vrf5007                               0s(0s*0)         3s(1s*3)          UP

  Yes                  0/RP0/CPU0
  2001:DB8:0000::8                          2001:DB8:FFF::8
  ```

```
vrf5008                             0s(0s*0)         3s(1s*3)         UP
```

- ```
  Router# show bfd session
  Tue Mar 26 19:32:00.554 UTC
  Src Addr            Dest Addr       VRF Name                      H/W NPU

                                        Local det time(int*mult)      State

                                      Echo             Async
  ------------------- --------------- ---------------- ----------------
  ----------
  209.165.201.1       10.0.0.1   vrf5001                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.2       10.0.0.2   vrf5002                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.3       10.0.0.3   vrf5003                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.4       10.0.0.4   vrf5004                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.5       10.0.0.5   vrf5005                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.6       10.0.0.6   vrf5006                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP

  209.165.201.7       10.0.0.7   vrf5007                     Yes 0/RP0/CPU0

                                      n/a              3s(1s*3)         UP
  ```

BFD on BVI interfaces

Explains how BFD operates on BVI in IRB environments to extend fast failure detection across bridged and routed domains using supported multipath session behavior.

Provides configuration and platform-specific considerations for implementing BFD on BVI interfaces.

Bidirectional forwarding detection on BVI

Describes how BFD extends fast path-failure detection to BVI-based IRB deployments by supporting multipath single-hop sessions over virtual interfaces.

Bidirectional forwarding detection on Bridge Group Virtual Interface (BVI) is a network protocol enhancement that

- detects path failures between routers in an integrated routing and bridging deployment
- provides low-overhead path failure detection, and
- supports multipath single-hop sessions over virtual interfaces or interfaces multiple hops away.
  ```

Table 52: Feature History Table

Feature Name	Release Information	Feature Description
Bidirectional forwarding detection on BVI	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*) *This feature is supported on: • 8711-48Z-M • 8712-MOD-M
Bidirectional forwarding detection on BVI	Release 25.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100]; Modular Systems (8800 [LC ASIC: P100]) This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM
Bidirectional forwarding detection on BVI	Release 7.10.1	Now you can extend the advantage of low-overhead and short-duration detection of path failures between routers to an Integrated Routing and Bridging (IRB) deployment scenario by configuring Bidirectional Forwarding Detection (BFD) on multipath single-hop sessions using Bridge-Group Virtual Interface (BVI). By configuring BFD on a multipath session, you can apply BFD over virtual interfaces or between interfaces that are multiple hops away.

BVI routing context

To establish a VLAN that extends across a router, the router actively forwards frames between interfaces while maintaining the integrity of the VLAN header. When configured for Layer 3 routing, the router terminates the VLAN and MAC layers at the arrival interface. BFD on BVI allows the maintenance of MAC layer header information by bridging the network layer protocol.

Role of IRB in VLAN spanning

VLANs segment a network by creating separate broadcast domains. To carry traffic for multiple VLANs over a single link, enable VLAN trunking. This capability lets the network extend VLAN connectivity across the link.

In VLAN deployments that include multiple routers, preserve the VLAN tags so the VLANs can extend across the network. Routers usually replace the VLAN header with a new header, whether they route or bridge the traffic. This behavior can prevent VLAN extension and reduce support for trunking and traffic bridging between VLANs.

Integrated Routing and Bridging (IRB) lets a router interface route and bridge traffic at the same time while preserving VLAN header information across routers. This behavior helps maintain VLAN extension in more complex networks.

Integrated routing and bridging

Use Integrated Routing and Bridging (IRB) to configure a router to route and bridge a network layer protocol on the same interface. This configuration preserves the VLAN header as the frame passes through the router. IRB uses a Bridge Group Virtual Interface (BVI) to route traffic between bridged and routed domains. The BVI is a virtual routed interface that represents the bridge group in the router. The BVI number matches the bridge group number.

To use the BVI as a routed interface, configure only Layer 3 attributes on it, such as network layer addresses. Do not configure Layer 3 attributes on interfaces that you use for protocol bridging.

BFD over IRB

BFD over IRB uses a Bridge Group Virtual Interface (BVI) to establish a multipath single-hop BFD session. In a BFD multipath session, BFD can monitor virtual interfaces and interfaces that are multiple hops away. This feature follows *RFC 5883, Bidirectional Forwarding Detection (BFD) for Multihop Paths.* It extends BFD low overhead and fast path-failure detection to IRB deployments.

BFD over BVI supports IPv4 addresses and IPv6 global addresses. This feature supports only asynchronous mode. Echo mode is not supported.

Restrictions of BFD over BVI

Provides protocol, interface, client, and hardware-mode limitations that apply to BFD over BVI deployments.

- BFD over BVI does not support IS-IS or OSPFv3 clients because these protocols require IPv6 link-local addresses, which this feature does not support.
- BFD over BVI does not support bundles or bundle subinterfaces. It supports only physical interfaces and physical subinterfaces.
- BFD over BVI does not support the HSRP client.
- On P100, K100, and A100 systems, BFD over BVI supports only two-pass mode. Single-pass mode is not supported, including Q200 compatibility mode.

Configure BFD over BVI

Shows how to configure multipath hosting, Layer 2 transport, BVI, bridge-domain, routing, and verification steps to enable BFD over BVI.

1. Configure the line cards to allow hosting of Multipath BFD sessions.

```
Router# configure
Router(config)# bfd
Router(config-bfd)# multipath include location 0/0/CPU0
Router(config-bfd)# multipath include location 0/2/CPU0
```

2. Configure the Layer 2 domain that includes the corresponding l2transport and BVIs.

```
Router(config)# interface HundredGigE0/0/0/6
Router(config-if)no shut
Router(config-if)exit
Router(config)# interface HundredGigE0/0/0/6.3001
Router(config-if)# l2transport
Router(config-if)# encapsulation dot1q 3001
```

```
Router(config-if)# rewrite ingress tag pop 1 symmetric
Router(config-if)# exit
```

3. Configure a BVI interface and assign an IP address.

   You can configure the BVI interface with either IPv4 or IPv6 address or both.

```
Router(config)# interface bvi3001
Router(config-if)# ipv4 address 192.168.1.1 255.255.255.0
Router(config-if)# exit
```

4. Configure BVI on the peer nodes.

```
Router(config-if)# l2vpn
Router(config-l2vpn)# bridge group bvi100
Router(config-l2vpn-bg)# bridge domain bfd-bvi
Router(config-l2vpn-bg-bd)# interface HundredGigE0/0/0/6.3001
Router(config-l2vpn-bg-bd-ac)# exit
Router(config-l2vpn-bg-bd)# routed interface bvi bvi3001
```

5. Configure OSPF as the routing protocol.

```
Router(config)# router ospf bfd-bvi
Router(config-ospf)# router-id 192.168.1.1
Router(config-ospf)# area 0
```

6. Configure BFD on BVI.

```
Router(config-ospf-ar)# interface bvi3001
Router(config-ospf-ar-if)# bfd fast-detect
Router(config-ospf-ar-if)# bfd multiplier 3
Router(config-ospf-ar-if)#commit
```

7. a) Verify the status of L2VPN bridge domain.

```
Router# show l2vpn bridge-domain brief
Mon Apr 24 11:31:04.314 UTC
Legend: pp = Partially Programmed.
Bridge Group:Bridge-Domain Name  ID  State  Num ACs/up  Num PWs/up  Num
PBBs/up Num VNIs/up
-------------------------------- --  -----  ----------- ---------
---------- ----------
bvi100:bfd-bvi                   0   up  2/2         0/0         0/0
   0/0
```

   b) Verify the status of OSPF connectivity. The creation of a BFD session requires the neighbor to be in the FULL/BDR state.

```
Router# show ospf neighbor
Mon Apr 24 11:40:41.900 UTC
* Indicates MADJ interface
# Indicates Neighbor awaiting BFD session up
Neighbors for OSPF bfd-bvi
Neighbor ID     Pri   State           Dead Time   Address         Interface
192.168.1.1     1     FULL/BDR         00:00:33    192.168.1.1     BVI3001
    Neighbor is up for 1d22h
Total neighbor count: 1
```

c) Verify the status of the BFD session. The output indicates the state of the BFD session is UP.

```
show bfd session interface bvi3001
Mon Apr 24 11:42:50.582 UTC
Interface   Dest Addr   Local det time(int*mult)   State  Echo  Async
H/W     NPU
---------  ------------ --------  --------------   -----  ----  -----
-----    -----
BVI3001     192.168.1.1  0s(0s*0)  750ms(250ms*3)   UP     Yes
0/2/CPU0
```

BFD over pseudowire headend (PWHE)

Describes how BFD supports PWHE deployments by providing rapid end-to-end fault detection for IPv4 and IPv6 static-route-based pseudowire services.

This section describes the configuration and usage of BFD over pseudowire headend (PWHE) interfaces to support service availability.

BFD over pseudowire headend

Describes how BFD over PWHE provides rapid end-to-end fault detection between CE and PWHE-PE links for IPv4 and IPv6 pseudowire services.

Bidirectional forwarding detection over pseudowire headend is a BFD feature that

- provides fast failure detection over customer edge to pseudowire headend provider edge links between eBGP neighbors
- supports BFD sessions per pseudowire for end-to-end fault detection between the customer edge and pseudowire headend provider edge, and
- supports BFDv4 for IPv4, BFDv6 for IPv6, and BFD asynchronous mode over PWHE and pseudowire VC types 4 and 5.

Table 53: Feature History Table

Feature Name	Release Information	Feature Description
BFD over pseudowire headend	Release 25.4.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100], 8700 [ASIC: K100])(select variants only*) *This feature is supported on: • 8011-32Y8L2H2FH • 8711-48Z-M
BFD over pseudowire headend	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I

Feature Name	Release Information	Feature Description
BFD over pseudowire headend	Release 24.3.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) You can now rapidly detect failures in pseudowires, minimizing downtime and ensuring service reliability. This feature continuously monitors the pseudowire end-to-end, providing quick responses to maintain the integrity of Layer 2 VPNs and Ethernet services. *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

BFD over PWHE supports:

- BFD sessions per pseudowire for end-to-end fault detection between the CE and the PWHE PE
- BFDv4 for IPv4 and BFDv6 for IPv6
- BFD asynchronous mode over PWHE, and
- Pseudowire VC type 4 and type 5.

Limitations of BFD over PWHE

Provides platform, routing, and session-type limitations for BFD over PWHE, including support only for static routes and single-hop IPv4 and IPv6 sessions.

BFD over PWHE feature is supported only on 88-LC1-12TH24FH-E line card.

Configure only these supported session types:

- Static routes for IPv4 and IPv6.
- Single-hop BFD sessions for IPv4 and IPv6.

Configure BFD over PWHE

Teaches you how to host BFD multipath sessions on a valid location, apply BFD fast detection to static IPv4 and IPv6 routes, and verify PWHE session status.

To enable BFD over PWHE, you must configure the specific line card to host BFD multipath sessions.

> **Note**
>
> For centralized platforms, only MPA locations, such as 0/1, are valid for BFD multipath configuration. Line card locations, such as 0/1/CPU0, are not supported and should not be used.

1. Configure the line card to host BFD multipath sessions.

   For distributed platforms:

```
Router# configure
Router(config)# bfd multipath include location 0/5/CPU0
```

   For centralized platforms:

```
Router# configure
Router(config)# bfd multipath include location 0/5
```

2. Configure BFD fast-detection capability on the specified IPv4 or IPv6 unicast destination address prefix and on the forwarding next-hop address.

```
Router# configure
Router(config)# router static
Router(config-static)# address-family ipv4 unicast
Router(config-static-afi)# 198.51.100.0/24 209.165.201.0 bfd fast-detect
minimum-interval 800 multiplier 3
Router(config-static-afi)# 198.51.100.0/24 209.165.202.0 bfd fast-detect
minimum-interval 800 multiplier 3
Router(config-static-afi)# exit
Router(config-static)# address-family ipv6 unicast
Router(config-static-afi)# 2001:DB8:C18:2:1::F/48
2001:DB8:D987:398:AE3:B39:333:783 bfd fast-detect minimum-interval 800
multiplier 3
Router(config-static-afi)# 2001:DB8:C18:2:1::F/48
2001:DB8:D987:398:AE3:B39:334:783 bfd fast-detect minimum-interval 800
multiplier 3
```

3. Verify the configuration.

   This example is for distributed platforms. For centralized platforms, change the value of location with MPA location.

```
Router# show bfd all session agent detail location 0/2/CPU0
IPv4:
-----
I/f: PW-Ether1002, Location: 0/2/CPU0
Dest: 209.165.202.0
Src: 209.165.201.0
 State: UP for 0d:0h:8m:17s, number of times UP: 1
 Session type: SW/V4/SH/PH
Received parameters:
 Version: 1, desired tx interval: 50 ms, required rx interval: 50 ms
 Required echo rx interval: 1 us, multiplier: 3, diag: None
 My discr: 2210464558, your discr: 4259847, state UP, D/F/P/C/A: 0/0/1/1/0
Transmitted parameters:
 Version: 1, desired tx interval: 50 ms, required rx interval: 50 ms
 Required echo rx interval: 0 ms, multiplier: 3, diag: None
 My discr: 4259847, your discr: 2210464558, state UP, D/F/P/C/A: 0/1/0/1/0
```

```
Timer Values:
 Local negotiated async tx interval: 50 ms
 Remote negotiated async tx interval: 50 ms
 Desired echo tx interval: 0 s, local negotiated echo tx interval: 0 ms
 Echo detection time: 0 ms(0 ms*3), async detection time: 150 ms(50 ms*3)
Label:
 Internal label: 24024/0x5dd8
Local Stats:
 Intervals between async packets:
   Tx: Number of intervals=100, min=46 ms, max=1804 ms, avg=342 ms
       Last packet transmitted 1250 ms ago
   Rx: Number of intervals=100, min=46 ms, max=1804 ms, avg=342 ms
       Last packet received 1250 ms ago
 Intervals between echo packets:
   Tx: Number of intervals=0, min=0 s, max=0 s, avg=0 s
       Last packet transmitted 0 s ago
   Rx: Number of intervals=0, min=0 s, max=0 s, avg=0 s
       Last packet received 0 s ago
 Latency of echo packets (time between tx and rx):
   Number of packets: 0, min=0 ms, max=0 ms, avg=0 ms
```

Additional BFD integrations

Summarizes additional BFD integrations and behaviors, including IPv6 operation in bundle manager domain and BGP strict-mode capabilities for improved interoperability.

This topic provides an overview of supplementary BFD functionalities and interoperability enhancements available for network configuration.

The BFD implementation includes support for IPv6 within Bundle Manager domains and introduces BGP BFD strict mode to enhance protocol interoperability and control.

BGP BFD strict-mode capabilities

Describes how BGP BFD strict-mode negotiation improves interoperability and stability by allowing BGP sessions to come up only after the corresponding BFD session is active and stable.

BGP BFD strict-mode is a configuration capability that

- makes BGP session establishment depend on BFD session state
- ensures BGP sessions start only when BFD sessions are active, and
- can enforce strict-mode requirements even when a peer does not natively support strict-mode.

- Strict-mode negotiate: Ensures BGP sessions start only when BFD sessions are active.
- Strict-mode negotiate override: Enforces strict-mode requirements even when a peer device does not natively support strict-mode.

Table 54: Feature History Table

Feature Name	Release Information	Feature Description
BGP BFD Strict-Mode Capabilities for Improved Interoperability	Release 25.4.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100])(select variants only*). * This feature is now supported on Cisco 8711-48Z-M routers.

Feature Name	Release Information	Feature Description
BGP BFD Strict-Mode Capabilities for Improved Interoperability	Release 25.1.1	Introduced in this release on: Fixed Systems (8700 [ASIC: K100], 8010 [ASIC: A100])(select variants only*) *This feature is supported on: • 8712-MOD-M • 8011-4G24Y4H-I

Feature Name	Release Information	Feature Description
BGP BFD Strict-Mode Capabilities for Improved Interoperability	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100])(select variants only*); Modular Systems (8800 [LC ASIC: P100])(select variants only*) You now have the ability to upgrade your network using Cisco's BGP BFD strict-mode negotiate and strict-mode negotiate override modes. These modes improve stability and cooperation between Cisco IOS XR and Cisco IOS XE devices. They ensure BGP sessions start only when BFD sessions are active. The override option enforces this even if a peer device does not support strict-mode. The feature ensures that a BGP session is established between neighbors only after the BFD session is established and stable. This ensures that the BFD functions as intended during a failure, promoting network stability and reliability. This feature introduces these changes: CLI: • The **bfd-state** keyword is introduced in the show bgp sessions command. • The BFDmode and BFDState fields are added to the show bgp neighbors command output. YANG Data Models: • New XPaths for `Cisco-IOS-XR-um-router-bgp-cfg.yang` (See GitHub, YANG Data Models Navigator) *This feature is supported on: • 8212-48FH-M • 8711-32FH-M • 88-LC1-36EH • 88-LC1-12TH24FH-E • 88-LC1-52Y8H-EM

BGP and BFD interoperability modes for routing resilience

Cisco provides two modes to improve BGP and BFD interoperability between Cisco IOS XR and Cisco IOS XE devices: BGP BFD strict-mode negotiate and BGP BFD strict-mode negotiate override. These modes bring up a BGP session only when the corresponding BFD session is up. The override mode continues to enforce this requirement even if the peer does not support strict mode. This behavior aligns with IETF standards and improves routing resilience.

Addressing interoperability challenges

Without these BGP BFD strict-mode capabilities, interoperability issues between Cisco IOS XR and Cisco IOS XE devices can prevent BGP sessions from coming up. These failures occur because of nonstandard strict-mode implementations. As a result, BGP routes might be advertised before the underlying BFD session is up, which can reduce routing stability.

BGP BFD strict-mode negotiate

BGP BFD strict-mode negotiate override

Devices advertise BFD strict-mode capability and check whether the peer also advertises it. If the peer advertises the capability, the devices operate in IETF mode and bring up BGP only when the corresponding BFD session is up. If the peer does not advertise strict-mode, the devices operate without enforcing strict mode. The **show bgp neighbor detail** command displays the capabilities sent and received.

Devices advertise BFD strict-mode capability and check whether the peer also advertises it. If the peer advertises the capability, the devices operate in IETF mode. If the peer does not advertise the capability, the devices continue to operate in IETF strict mode and wait for BFD to come up before moving past the Open BFD state. If the peer resets the session because it does not support strict-mode, the device logs the event for that neighbor, stops advertising the capability in later Open messages, and continues to operate in IETF strict mode.

Verify peer advertisement for BFD strict-mode capability

Describes the verification process for BFD strict-mode negotiate capability and negotiate override mode to ensure BGP session stability.

This topic provides the logic for verifying peer advertisement of strict mode capability and the resulting device behavior for BGP session establishment.

Table 55: BFD strict-mode negotiate capability

If	Then
The peer advertises strict mode capability	The device brings up the BGP sessions only if the BFD session is up; this is in compliance with the IETF standard.
The peer does not advertise strict mode capability	The device starts operation without enforcing any strict mode.

Table 56: BFD strict-mode negotiate override mode

If	Then
The peer advertises strict mode capability	The device brings up the BGP sessions only if the BFD session is up.
The peer advertises that it does not support strict mode capability	The device overrides the neighbor's capability notification and start operation in strict mode.

Associated RFCs

Lists the RFC documents that define the standards and specifications for BFD.

This table lists the RFCs associated with BFD, including their titles and publication dates.

RFCs	Title
rfc5880_bfd_base	*Bidirectional Forwarding Detection* , June 2010
rfc5881_bfd_ipv4_ipv6	*BFD for IPv4 and IPv6 (Single Hop)* , June 2010

Technical assistance

Lists Cisco Technical Support resources for technical content, product information, and tools.

Use this section to access Cisco Technical Support documentation and resources.

Description	Link
Use the Cisco Technical Support website to find searchable technical content, including products, technologies, solutions, technical tips, and tools. Registered Cisco.com users can sign in for additional content.	http://www.cisco.com/techsupport

10 Implementing Fast Reroute Loop-Free Alternate

Topics:

Explains how LFA-based fast reroute provides rapid traffic recovery through local and remote backup paths, repair path selection, and protocol support in IS-IS and OSPF.

This chapter describes how LFA-based fast reroute operates, how backup paths are calculated, and how routing protocols such as IS-IS and OSPF support local and remote LFA implementations, including repair path selection and loop prevention.

Fast reroute with local loop-free alternate

Explains how loop-free alternate fast reroute protects traffic by calculating backup paths and redirecting packets quickly around failed links or nodes through remote alternates.

Fast reroute loop-free alternate is a network recovery feature that

- calculates backup paths to protect against link or node failures
- enables rapid traffic redirection to a remote loop-free alternate, and
- supports multi-hop protection while integrating with routing protocols during topology changes.

Prerequisites for IPv4/IPv6 loop-free alternate fast reroute

Loop-Free Alternate (LFA) Fast Reroute (FRR) can protect paths that are reachable through an interface only if the interface is a point-to-point interface.

Restrictions for loop-free alternate fast reroute

Provides implementation limits for LFA FRR, including MPLS, topology, interface, and tunnel restrictions that affect backup path computation.

Configure the remote LFA backup path for MPLS traffic by using only LDP.

Restrict LFA calculations to interfaces or links in the same level or area.

- Excluding all neighbors on the same LAN during backup LFA computation can make repair paths unavailable in some topologies.

Protect only physical interfaces. Subinterfaces, tunnels, and virtual interfaces are not supported.

IPv4 and IPv6 LFA FRR over TE tunnels are not supported.

Loop-free alternate as a backup path

Describes how an LFA provides a backup path that redirects traffic after failures without creating loops or depending on the failed element.

A loop-free alternate (LFA) provides a backup path that redirects traffic after a network failure. It sends traffic to a node other than the primary neighbor so that forwarding can continue during the transition.

An LFA makes the forwarding decision without knowing the exact point of failure. The backup path must not use the failed element, route traffic through the protecting node in a way that creates dependency on the failure, or create a forwarding loop. By default, LFA is enabled on all supported interfaces that can serve as a primary path.

Per-prefix LFAs provide these advantages:

- The repair path forwards traffic while the primary link is down.
- All destinations that have a per-prefix LFA remain protected. Only some destinations beyond the failure can remain unprotected.

How repair paths and LFA calculation maintain traffic flow

Describes how repair paths and LFA calculations precompute alternate forwarding paths to reduce packet loss and maintain connectivity until routing convergence completes.

Repair paths and loop-free alternate (LFA) calculation help maintain traffic flow during routing transitions. They reduce packet loss by precomputing alternate paths that can be used after a link or router failure.

Repair paths forward traffic while the network converges. When a link or router fails because of a physical-layer signal loss, only the adjacent routers detect the failure immediately. Other routers remain unaware until the routing protocol

propagates the update, which can take several hundred milliseconds. During this interval, traffic must be redirected along an alternate path.

This process uses these components:

- Adjacent router: Uses repair paths for packets that would otherwise traverse the failed link until routing convergence is complete.
- Repair paths: Are precomputed so that the router can activate them as soon as it detects a failure.

These stages describe the implementation of repair paths and LFA calculation.

1. The LFA FRR feature uses specific repair paths to maintain connectivity:
   - Equal-cost multipath (ECMP): Uses another member of an equal-cost path set as an alternate path when one link fails.
   - Loop-free alternate (LFA): Acts a next hop that can reach the destination without sending the packet into a loop. Downstream paths are a subset of LFAs.
2. IS-IS calculates LFAs based on *RFC 5286*, with modifications that optimize memory usage.
   - IS-IS performs shortest path first (SPF) calculation for each neighbor.
   - IS-IS evaluates prefixes for each result.
   - IS-IS retains only the best repair path, so it does not need to store SPF results for all neighbors.

After routing convergence completes, all routers update their forwarding information, and the failed link is removed from the routing computation.

How tiebreaking selects repair paths for candidate LFAs

Describes how routing protocols apply tiebreaking rules to select repair paths and choose one loop-free alternate per primary path for efficient failover.

A routing protocol computes repair paths for prefixes by applying tiebreaking rules. The result is a set of prefixes with primary paths, where some primary paths also have repair paths.

A routing protocol computes repair paths for prefixes by applying tiebreaking rules to candidate LFAs. This process removes extra candidates and selects one LFA for each primary path and prefix.

These stages describe the tiebreaking process for candidate LFAs.

1. Identify candidate LFAs.

   The routing protocol starts with a set of candidate LFAs for prefixes that have primary paths. Some of these primary paths can also have repair paths.
2. The routing protocol evaluates candidate LFAs against specific attributes to narrow down the selection.

   The routing protocol applies default tiebreaking rules to narrow the set of candidate LFAs. You can modify these rules if needed.
   - Downstream: Eliminates candidates whose metric to the protected destination is lower than the protecting node's metric to that destination.
   - Linecard-disjoint: Eliminates candidates that share the same line card as the protected path.
   - Shared Risk Link Group (SRLG): Eliminates candidates that belong to the same SRLG as the protected path.
   - Load-sharing: Distributes the remaining candidates among prefixes that share the protected path.
   - Lowest-repair-path-metric: Eliminates candidates whose metric to the protected prefix is higher.
   - Node protecting: Eliminates candidates that do not provide node protection.

- Primary-path: Eliminates candidates that are not equal-cost multipath (ECMP) paths.
- Secondary-path: Eliminates candidates that are ECMP paths.

3. Skip any rule that removes all candidates.

   If a tiebreaking rule would eliminate all candidate LFAs, the routing protocol skips that rule.

   At least one candidate LFA remains available for selection.

4. Select the repair path.

   After evaluating the candidate LFAs, the routing protocol selects one LFA for each primary path and prefix.

   The routing protocol assigns one repair path to each eligible primary path and prefix to help distribute traffic if the primary path fails.

The routing protocol selects a single repair path for each eligible primary path and prefix, which helps maintain traffic flow after a primary path failure.

IS-IS support for IP fast reroute

Describes how IS-IS uses IP fast reroute to precompute loop-free alternate next hops and reduce traffic loss while new primary routes are recomputed and installed.

IP fast reroute (FRR) is a network resiliency feature that

- lets IS-IS compute loop-free alternate (LFA) next-hop routes for use when the primary path fails
- computes LFA next-hop routes for each prefix
- appliess tiebreaking rules to select one LFA when multiple candidates exist, and
- distributes prefixes evenly across available LFA paths.

IS-IS failure recovery process

When a local link fails, IS-IS performs this recovery process:

1. Recomputes the primary next-hop routes for all affected prefixes.
2. Updates the routing information base (RIB) and forwarding information base (FIB) with the new routes.

> **Warning**
>
> Without IP FRR, traffic to affected prefixes is dropped until the forwarding plane is updated. This update can take several hundred milliseconds.

Configure FRR with local LFA using IS-IS

Teaches you how to configure IS-IS for IPv4 unicast and enable per-prefix local LFA fast reroute on an interface for link protection.

Use this task to configure the IS-IS routing process for IPv4 unicast and enable per-prefix fast reroute on an interface.

This example uses IS-IS process `ring`, Loopback0, and HundredGigE `0/2/0/0`. Replace these sample values with values from your environment.

This example shows an IPv4 LFA FRR configuration. IPv6 LFA FRR is also supported.

1. Enter IS-IS router configuration mode and configure the IS-IS process attributes.

```
Router# configure
Router(config)# router isis ring
Router(config-isis)# is-type level-1
Router(config-isis)# net 49.0001.0000.0000.0007.00
Router(config-isis)# nsf cisco
Router(config-isis)# address-family ipv4 unicast
Router(config-isis-af)# metric-style wide
```

2. Configure Loopback0 for IPv4 unicast under the IS-IS process.

```
Router(config-isis-af)# exit
Router(config-isis)# interface Loopback0
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
```

3. Configure the physical interface and enable per-prefix fast reroute.

```
Router(config-isis-if)# exit
Router(config-isis)# interface HundredGigE 0/2/0/0
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if-af)# fast-reroute per-prefix
Router(config-isis-if-af)# commit
```

   The router is configured for IS-IS IPv4 unicast, and per-prefix fast reroute is enabled on the interface.

IS-IS fast reroute is configured on the router and enabled on the specified interface.

Configure OSPF per-prefix local LFA fast reroute

Teaches you how to configure OSPF for IPv4 unicast and enable per-prefix local LFA fast reroute on a specific interface.

Use this task to configure OSPF and enable per-prefix local LFA fast reroute on an interface.

This example uses OSPF process `50`, router ID `10.1.1.1`, area `0`, Loopback0, and HundredGigE `0/2/0/0`. Replace these sample values with values from your environment.

This example shows an IPv4 LFA FRR configuration. IPv6 LFA FRR is also supported.

1. Enter OSPF router configuration mode and configure the OSPF process for IPv4 unicast.

```
Router# configure
Router(config)# router ospf 50
Router(config-ospf)# router-id 10.1.1.1
Router(config-ospf)# address-family ipv4 unicast
Router(config-ospf-af)# area 0
```

2. Add Loopback0 to OSPF area 0.

```
Router(config-ospf-af-area)# interface Loopback0
```

3. Add HundredGigE `0/2/0/0` to OSPF area 0 and enable per-prefix local LFA fast reroute.

```
Router(config-ospf-af-area)# interface HundredGigE 0/2/0/0
Router(config-ospf-af-area-if)# fast-reroute per-prefix
Router(config-ospf-af-area-if)# commit
```

   OSPF is configured for IPv4 unicast, and per-prefix local LFA fast reroute is enabled on the interface.

The router is configured for local LFA fast reroute with OSPF on the specified interface.

FRR recirculation avoidance

Explains how the Fast Reroute recirculation avoidance mechanism eliminates packet recirculation during failover events, and describes its benefits for PE router deployments by improving bandwidth efficiency, preserving line-rate throughput, and reducing congestion and traffic loss during successive FRR events.

Fast Reroute (FRR) recirculation avoidance mechanism is a core routing enhancement that

- eliminates packet recirculation during FRR events,
- improves bandwidth efficiency and prevents traffic loss in PE edge router deployments, and
- maintains line-rate throughput by optimizing the forwarding chain during failover scenarios.

Table 57: Feature History Table

Feature Name	Release Information	Feature Description
Fast Reroute recirculation avoidance	Release 26.2.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200]); Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC:Q200]) You can now eliminate packet recirculation during Fast Reroute events in PE routers. The feature ensures that traffic switches to backup paths without requiring recycle operations during both first and subsequent FRR events. Previously, traffic switched to backup paths during FRR events that required one or more recycle passes. This increased bandwidth and reducing efficiency.

FRR recirculation avoidance eliminates packet recirculation during FRR events in PE edge router deployments, improving bandwidth efficiency and reducing traffic loss.

Previously, when the primary transit path failed during the first FRR event, the router redirected traffic to a backup path that required one packet recycle. If a second FRR event occurred, traffic was redirected to another backup path that required an additional recycle. These recirculation steps consumed NPU bandwidth and, under high load conditions, could lead to congestion and potential traffic drops due to limited recycle port capacity.

Benefits of FRR recirculation avoidance

Describes the benefits of FRR recirculation avoidance, including preserved bandwidth, improved forwarding stability on heavily loaded NPUs, and maintained line-rate throughput during failover by removing recycle-path dependency from the forwarding chain.

The benefits of FRR recirculation avoidance include:

- Bandwidth preservation: By removing the recycle path, the feature prevents the heavy bandwidth impact associated with FRR events.
- Stability: Reduces the risk of traffic drops on fully loaded NPUs.
- Performance: Maintains line-rate throughput during FRR scenarios by optimizing the forwarding chain.

Limitations of FRR recirculation avoidance

Describes the limitation of FRR recirculation avoidance and explains that the feature is supported only on the gRIBI forwarding path.

FRR recirculation avoidance is supported exclusively on the gRIBI forwarding path. Deployments utilizing other forwarding paths are not supported for this feature.

Fast reroute with remote loop-free alternate

Describes how remote LFA tunnels traffic to a precomputed backup node more than one hop away to minimize loss during failures and speed recovery.

The fast reroute with remote loop-free alternate (FRR Remote LFA) is a resiliency feature that

- enables rapid traffic redirection during link or router failures
- uses precomputed alternate next hops to bypass failed network segments, and
- supports recovery for failures that are more than one hop away from the point of local repair.

Table 58: Feature History Table

Feature Name	Release Information	Feature Description
Fast Reroute with Remote Loop-Free Alternate	Release 25.1.1	Introduced in this release on: Fixed Systems (8010 [ASIC: A100])(select variants only*) *This feature is supported on Cisco 8011-4G24Y4H-I routers.
Fast Reroute with Remote Loop-Free Alternate	Release 24.4.1	Introduced in this release on: Fixed Systems (8200 [ASIC: P100], 8700 [ASIC: P100, K100])(select variants only); Modular Systems (8800 [LC ASIC: P100])(select variants only*) This feature minimizes traffic loss by rerouting packets around failed links quickly. It precomputes repair paths using the IS-IS routing protocol, allowing routers to switch immediately to these paths when a failure occurs, reducing transition time to under 50 milliseconds. *This feature is supported on: - 8212-48FH-M - 8711-32FH-M - 8712-MOD-M - 88-LC1-36EH - 88-LC1-12TH24FH-E - 88-LC1-52Y8H-EM

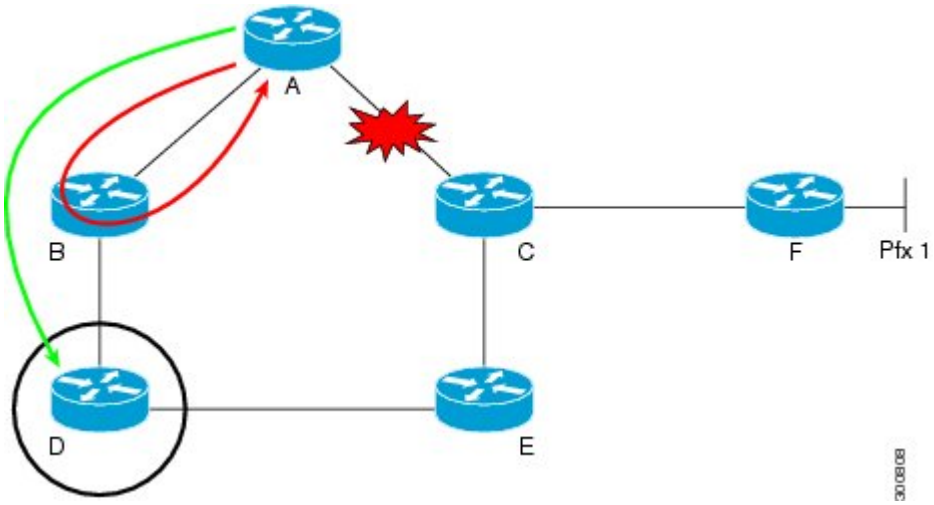
Routing transition and convergence

When a link or router fails, distributed routing protocols must compute new paths. During this routing transition, traffic to affected destinations can be interrupted until the routers converge. Standard IGP or BGP convergence can take several hundred milliseconds. FRR remote LFA reduces this interruption by letting the router switch immediately to a precomputed repair path when it detects the failure.

FRR remote LFA with ring topology

Consider this ring topology for FRR remote LFA.

Figure 5: FRR remote LFA with ring topology



Device A sends traffic for destination F through next-hop B. Device B cannot serve as a loop-free alternate (LFA) for prefixes that nodes C and F advertise. The actual LFA is node D, but node D is not directly connected to the protecting node, A. To protect prefixes that node C advertises, node A must tunnel the packet around the failed A-to-C link to node D, as long as the tunnel does not use the failed link.

FRR remote LFA lets you tunnel a packet around a failed link to a remote loop-free alternate that is more than one hop away. The feature automatically creates the tunnel to avoid looping.

How FRR remote LFA computes and installs repair paths

Describes how Remote LFA identifies a remote backup next hop, builds labeled transport paths, and installs repair entries so traffic can switch immediately after a failure.

FRR remote LFA protects network topologies in which local LFA fast reroute (FRR) does not provide enough coverage, such as ring-based topologies. It lets interior gateway protocols (IGPs) use non-directly connected neighbors as backup paths.

This process uses these components:

- IGP: Identifies a non-directly connected neighbor as the LFA backup path.
- Label Distribution Protocol (LDP): Establishes a labeled backup label-switched path (LSP) to the remote next hop and a transport LSP for tunneling.
- Cisco Express Forwarding: Installs the repair paths that the routing protocols compute.

These stages describe how FRR remote LFA works.

1. IGP identifies a remote neighbor that is more than one hop away as the LFA backup path.
2. LDP establishes a labeled backup LSP to the remote next hop.
3. LDP creates a transport LSP that tunnels traffic to the remote node.

4. Cisco Express Forwarding installs the repair path so that the router can switch traffic immediately when it detects a link or router failure.

The network can achieve sub-50 millisecond convergence by using a precomputed repair path.

Configure remote FRR with remote LFA using IS-IS

Teaches you how to configure IS-IS Remote LFA for per-prefix or per-link protection and enable MPLS LDP tunneling for remote repair paths.

This example shows an IPv4 LFA FRR configuration. IPv6 LFA FRR is also supported.

1. Configure the IS-IS routing process and interfaces for remote LFA.

```
Router# configure
Router(config)# router isis ring
Router(config-isis)# is-type level-1
Router(config-isis)# net 49.0001.0000.0000.0007.00
Router(config-isis)# nsf cisco
Router(config-isis)# address-family ipv4 unicast
Router(config-isis-af)# metric-style wide
Router(config-isis-af)# exit
Router(config-isis)# interface Loopback 0
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
Router(config-isis-if)# exit
Router(config-isis)# interface TenGigabitEthernet 0/0/0/4
Router(config-isis-if)# point-to-point
Router(config-isis-if)# address-family ipv4 unicast
```

2. Enable FRR LFA computation.

If...	Then...
Enable FRR LFA prefix dependent computation	Configure the `fast-reroute per-prefix` command.
Enable FRR LFA prefix independent per-link computation	Configure the `fast-reroute per-link` command.

```
Router(config-isis-af)# fast-reroute per-prefix
Router(config-isis-af)# fast-reroute per-prefix remote-lfa tunnel mpls-ldp
Router(config-isis-af)# fast-reroute per-prefix remote-lfa prefix-list
Router(config-isis-af)# commit
```

FRR with remote LFA running configuration

```
router isis ring
 is-type level-1
 net 49.0001.0000.0000.0007.00
 nsf cisco
 address-family ipv4 unicast
 fast-reroute per-prefix remote-lfa prefix-list abc
  metric-style wide
 !
 interface Loopback0
  point-to-point
  address-family ipv4 unicast
 !
 !
```

```
 interface TenGigabitEthernet 0/0/0/4
  point-to-point
  address-family ipv4 unicast
   fast-reroute per-prefix remote-lfa prefix-list
   fast-reroute per-prefix remote-lfa tunnel mpls-ldp
```

3. Verify the route summary information about the specified routing table.

```
Router# show route 10.3.3.3
Routing entry for 10.3.3.3/32
   Known via "isis 44", distance 115, metric 20, type level-1
   Installed Nov 15 19:43:13.367 for 00:00:34
   Routing Descriptor Blocks
     10.1.1.1, from 10.3.3.3, via TenGigE0/0/0/0, Backup (remote)
       Remote LFA is 10.9.9.9
      Route metric is 0
     10.1.1.2, from 10.3.3.3, via TenGigE0/7/0/3, Protected
       Route metric is 20
   No advertising protos.
```

4. Verify the MPLS LDP configuration.

```
Router# show running mpls ldp
Codes:
   - = GR label recovering, (!) = LFA FRR pure backup path
   {} = Label stack with multi-line output for a routing path
   G = GR, S = Stale, R = Remote LFA FRR backup

 Prefix           Label   Label(s)       Outgoing     Next Hop
Flags
                  In      Out            Interface                       G
S R
 ------------- ----- -------- -------- --------------- ----
 192.0.2.0/24     16019   { 16001        Te0/0/0/0    10.1.1.1        (!)
  R
                            28006 }                   (10.9.9.9)
                  ImpNull        Te0/7/0/3    192.0.2.1
 192.0.2.1/32     16013   ImpNull        Te0/0/0/0    10.1.1.1
 192.0.1.0/32     16014   { 16001        Te0/0/0/0    10.1.1.1        (!)
  R
                            16002 }                   (10.9.9.9)
                  ImpNull        Te0/7/0/3    192.0.2.2
 10.9.9.9/32      16012   16001          Te0/0/0/0    10.1.1.1
                  28006          Te0/7/0/3    192.0.2.1
 10.23.1.0/24     16018   16004          Te0/0/0/0    10.1.1.1         (!)
                  ImpNull        Te0/7/0/3    192.0.2.1
 10.34.1.0/24     16015   ImpNull        Te0/0/0/0    10.1.1.1
 10.0.0.1/32      16011  { 16001         Te0/0/0/0     10.1.1.1         (!)
   R
                            16013 }                   (10.9.9.9)
                  16016          Te0/7/0/3    192.0.2.1
 10.100.0.2/32     16010   { 16001       Te0/0/0/0    10.1.1.1         (!)
   R
```

Configure FRR with remote LFA using OSPF

Teaches you how to configure OSPF Remote LFA for IPv4 traffic protection by enabling per-prefix or per-link fast reroute with MPLS LDP tunneling.

This example shows an IPv4 LFA FRR configuration. IPv6 LFA FRR is also supported.

1. Enter configuration mode.

```
Router# configure
```

2. Enter OSPF configuration mode and specify the process ID.

```
Router(config)# router ospf 50
Router(config-ospf)# router-id 10.1.1.1
```

3. Enter address-family configuration mode.

```
Router(config-ospf)# address-family ipv4 unicast
Router(config-ospf-af)# area 0
```

4. Configure the interface for OSPF.

```
Router(config-ospf-af)# interface Loopback 0
Router(config-ospf-af)# exit
```

5. Configure the interface for FRR LFA.

If...	Then...
Enable FRR LFA prefix dependent computation	Use the **fast-reroute per-prefix** command.
Enable FRR LFA prefix independent per-link computation	Use the **fast-reroute per-link** command.

```
Router(config-ospf)# interface HundredGigE0/2/0/0
Router(config-ospf-if)# fast-reroute per-prefix remote-lfa tunnel mpls-ldp
Router(config-ospf-if)# commit
```

IPv4 LFA FRR running configuration

```
router ospf 50
 router-id 10.1.1.1
address-family ipv4 unicast
 area 0
   interface Loopback0
  !
  interface HundredGigE 0/2/0/0
    fast-reroute per-prefix remote-lfa tunnel mpls-ldp
  !
!
```

11 Flexible Algorithm in IP Networks

Topics:

Explains how IGP flexible algorithm computes constraint-based IPv4 and IPv6 paths so different traffic types can follow customized routes instead of the default least-cost path.

IGP flexible algorithm is a routing capability that

- lets routers calculate paths by using a defined combination of calculation type, metric type, and constraints instead of the default shortest-path calculation
- enables computation of constraint-based paths that meet specific network requirements, and
- supports path computation for both IPv4 and IPv6 destination addresses.

Operational behavior of IGP flexible algorithm

When a packet is sent to an IP address associated with a flexible algorithm, the packet follows the constraint-based path computed for that algorithm. If the destination IP address is not associated with a flexible algorithm, the packet follows the regular IGP least-cost path to the egress node.

Flexible algorithm supports various traffic management scenarios:

- Traffic segregation: Different traffic types can be forwarded over different paths.
- Performance optimization: Voice traffic can be routed over a low-latency path while other traffic uses the regular IGP path.

In IS-IS, protocol extensions support the use of flexible algorithm for IP networks.

Flexible algorithm definition

Describes how a flexible algorithm definition combines calculation type, metric type, and constraints, and explains why all participating routers must use an identical definition for consistent forwarding.

A flexible algorithm definition is a set that

- includes a calculation type
- includes a metric type, and
- includes a set of constraints.

Flexible algorithm identification and consistency

A flexible algorithm uses a numeric identifier in the range 128 to 255. This identifier is provisioned and associated with a specific flexible algorithm definition.

Routers use the flexible algorithm definition to calculate paths in a consistent way for a given flexible algorithm. To ensure loop-free forwarding, all routers that participate in the same flexible algorithm and that are within the same flexible algorithm definition advertisement scope must use an identical definition for that flexible algorithm.

A flexible algorithm definition is considered identical only when the participating routers agree on all of the following elements:

- Calculation type
- Metric type
- Constraints

If routers do not use the same flexible algorithm definition for a given flexible algorithm identifier, the computed paths may be inconsistent across the network. Consistent definition and advertisement of the flexible algorithm are therefore required for correct forwarding behavior.

This behavior is defined in `ietf-lsr-flex-algo`.

Flexible algorithm definition advertisement

Describes how routers advertise a common flexible algorithm definition, use advertisement priority to select one consistent definition, and require a valid advertisement for path computation and forwarding.

Flexible algorithm definition advertisement is the process of advertising a flexible algorithm definition that

- ensures participating routers use the same path-computation rules for a flexible algorithm
- includes attributes such as metric type, affinity constraints, and excludes SRLG constraints, and
- requires at least one router in the area to advertise a valid definition for the flexible algorithm to operate.

Flexible algorithm advertisement requirements

Use the **advertise-definition** command to enable a router to advertise the definition for a particular flexible algorithm.

The advertisement process has these requirements:

- At least one router in the area must advertise the flexible algorithm definition.
- Two advertising routers are recommended for redundancy.
- The advertisement includes a priority value so that all routers can select a single, consistent definition.
- A flexible algorithm does not operate unless a valid definition is advertised.

IP flexible algorithm prefix advertisement

Provides information about how IPv4 and IPv6 prefix reachability is advertised for a flexible algorithm.

The IPv4 and IPv6 Algorithm Prefix Reachability TLVs defined in *draft-ietf-lsr-ip-flexalgo* advertise prefix reachability for a flexible algorithm.

These TLVs identify IP prefixes that are associated with a specific flexible algorithm so that routers can calculate and use the corresponding algorithm-specific forwarding path.

IP flexible algorithm participation

Describes how applications signal participation in a flexible algorithm and notes that IP flexible algorithm uses the IS-IS IP Algorithm Sub-TLV for this purpose.

Each application that uses flexible algorithm requires its own participation signaling.

IP flexible algorithm uses the IS-IS IP Algorithm Sub-TLV specified in *draft-ietf-lsr-ip-flexalgo* to signal participation.

How IP flexible algorithm paths are computed

Describes how the router processes algorithm prefix advertisements, maintains separate path sets per data plane and flexible algorithm, and computes primary and backup paths with added CPU cost.

This process applies when a router participates in IP flexible algorithm path computation.

The router maintains a separate set of paths for each flexible algorithm and for each data plane. The computation uses only IP flexible algorithm prefixes that are advertised in the IPv4 and IPv6 Algorithm Prefix Reachability TLVs.

These stages describe how the router computes IP flexible algorithm paths.

1. The router receives and processes the IPv4 and IPv6 Algorithm Prefix Reachability TLVs for IP flexible algorithm prefixes.
2. The router selects only the advertised IP flexible algorithm prefixes that apply to the flexible algorithm being computed.
3. The router maintains a separate set of paths for the flexible algorithm for each data plane.
4. The router calculates the primary and backup paths for the selected IP flexible algorithm prefixes.
5. The router advertises the IPv4 or IPv6 Algorithm Prefix Reachability TLVs that are required for IP flexible algorithm operation.

The router computes algorithm-specific IP paths for each data plane. The processing impact is proportional to the number of IP flexible algorithms on the participating router and can require additional CPU cycles.

IP flexible algorithm forwarding

Describes the forwarding mechanism for IP Flexible Algorithm, which utilizes base IPv4 and IPv6 packets to route traffic based on specific algorithm constraints. This process ensures consistent path calculation and protection across the network topology.

IP flexible algorithm forwarding is the IP forwarding behavior that

- uses base IPv4 and IPv6 packets to forward traffic
- installs IP flexible algorithm prefixes that are advertised by participating routers in the associated topology and algorithm, and
- forwards packets by using the path that is computed for the associated flexible algorithm.

IP flexible algorithm prefixes that are associated with a flexible algorithm can be protected by local loop-free alternates (LFAs) and topology-independent loop-free alternates (TI-LFAs).

When a prefix is associated with a flexible algorithm, the router calculates the local LFA path for that prefix by using the same flexible algorithm and topology that are used for the primary path. This behavior ensures that the backup path follows the same constraints as the primary path.

For more information about protecting IP flexible algorithm prefixes, see Protecting IP flexible algorithm prefixes on page 216.

Configure IP flexible algorithm

Teaches you how to enable IP data-plane participation, define and advertise a flexible algorithm, configure affinity maps, and associate interfaces and IP addresses with the algorithm.

Use this task to configure IP flexible algorithm on the native IP data plane in IS-IS.

This workflow enables IP flexible algorithm by configuring participation, defining the flexible algorithm definition, advertising the definition, and configuring affinity mapping.

To complete the full workflow, also verify the configuration and associate the interface IP address with the flexible algorithm.

Before you begin, determine the flexible algorithm number, metric type, priority value, affinity names, and affinity bit positions that you want to use in the network.

1. Enable flexible algorithm participation on the native IP data plane.

```
Router(config)# router isis instance
Router(config-isis)# flex-algo algo
Router(config-isis-flex-algo)# data-plane ip
```

> **Note**
>
> Segment Routing is the default data plane. To use the IP data plane, enable the **data-plane ip** command.

2. Define the flexible algorithm parameters.

```
Router(config)# router isis instance
Router(config-isis)# flex-algo algo
Router(config-isis-flex-algo)# metric-type {delay | te}
Router(config-isis-flex-algo)# affinity {include-any | include-all |
exclude-any} name1, name2
Router(config-isis-flex-algo)# priority priority-value
```

   - *name1* and *name2* specify affinity-map names.
   - *priority-value* specifies the priority that is used during flexible algorithm definition election.

3. Advertise the flexible algorithm definition.

```
Router(config)# router isis instance
Router(config-isis)# flex-algo algo
Router(config-isis-flex-algo)# advertise-definition
```

4. Define the affinity map.

```
Router(config)# router isis instance
Router(config-isis)# affinity-map name bit-position bit-number
```

- *name* specifies the affinity-map name.
- *bit-number* specifies the bit position in the Extended Admin Group bitmask.

5. Associate the flexible algorithm-specific affinities with an interface.

```
Router(config)# router isis instance
Router(config-isis)# interface type interface-path-id
Router(config-isis-if)# affinity flex-algo name1, name2
```

name1 and *name2* specify the affinity-map names to associate with the interface.

The router is configured to participate in IP flexible algorithm on the native IP data plane, use the defined flexible algorithm parameters, advertise the flexible algorithm definition, and apply affinity mapping.

Verify the flexible algorithm configuration and associate the interface IP address with the flexible algorithm.

Associate an IP address with a flexible algorithm

Teaches you how to associate an interface IPv4 or IPv6 address with a specific flexible algorithm or use the default algorithm when none is specified.

Use this task to associate a flexible algorithm with an interface global IPv4 or IPv6 address.

You can associate an algorithm with an IPv4 or IPv6 address on an interface by using the **address** command with the *algorithm* option.

> **Note**
>
> If you do not associate an algorithm, the default algorithm value of *0* is used.

1. Enter interface configuration mode for the interface whose address you want to associate with a flexible algorithm.

```
Router(config)# interface loopback 1
```

2. Configure the IPv4 or IPv6 address with the flexible algorithm.

```
Router(config-if)# ipv4 address 192.0.2.1/32 algorithm 128
Router(config-if)# ipv6 address 2001:DB8::1/64 algorithm 128
Router(config-if)# commit
```

- For IPv4, use **ipv4 address prefix-length [secondary | algorithm algo-no]**.
- For IPv6, use **ipv6 address prefix-length algorithm algo-no**.

The interface IP address is associated with the specified flexible algorithm.

Associating an IPv4 address with a flexible algorithm

```
Router(config)# interface loopback 1
Router(config-if)# ipv4 address 192.0.2.1/32
Router(config-if)# ipv4 address 192.0.2.1/32 algorithm 128
Router(config-if)# commit
```

Associating an IPv6 address with a flexible algorithm

```
Router(config)# interface loopback 0
Router(config-if)# ipv6 address 2001:DB8::1/64
Router(config-if)# ipv6 address 2001:DB8::1/64 algorithm 128
Router(config-if)# commit
```

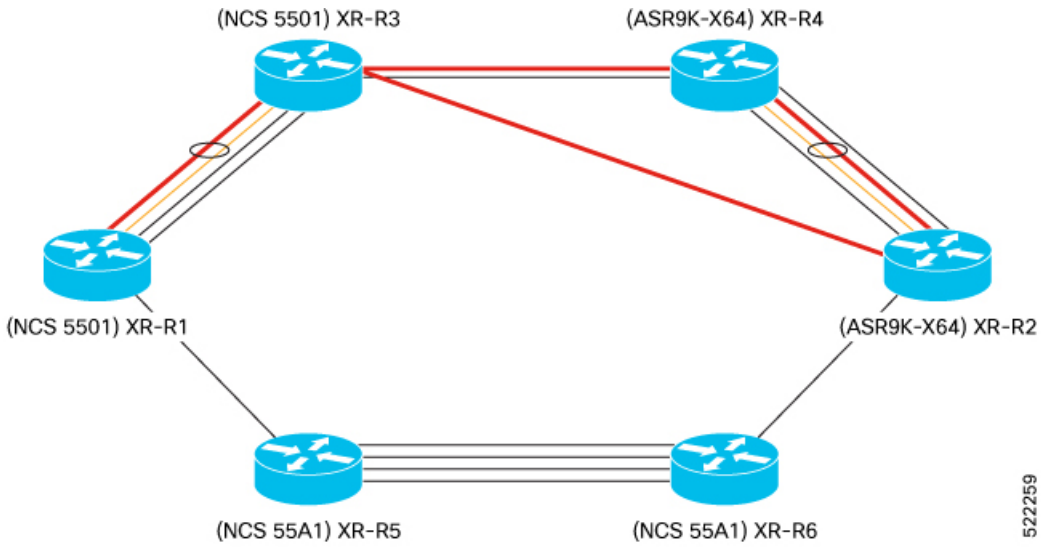
Configure IS-IS IP flexible algorithm

Shows how to configure IS-IS IP Flexible Algorithm by defining the algorithm, advertising the definition, mapping affinities, and associating IPv4 and IPv6 addresses with the selected algorithm.

Use this task to configure IS-IS IP flexible algorithm on a router and associate IPv4 and IPv6 interface addresses with the configured algorithm.

This example uses a six-node topology from R1 to R6. In this topology, node R2 defines the flexible algorithm definition, and the IP flexible algorithm path uses nodes R2, R4, R3, and R1.

Figure 6:



This example uses flexible algorithm 151 and associates that algorithm with the IPv4 and IPv6 addresses on Loopback51.

Before you begin, determine the IS-IS instance, flexible algorithm number, metric type, affinity-map name, and the interface addresses that you want to associate with the flexible algorithm.

1. Configure the IS-IS process, define the affinity map, and configure the flexible algorithm definition.

```
Router(config)# router isis 1
Router(config-isis)# is-type level-2-only
Router(config-isis)# net 49.0000.0000.0000.0012.00
Router(config-isis)# affinity-map RED bit-position 4
Router(config-isis)# flex-algo 151
Router(config-isis-flex-algo)# data-plane ip
Router(config-isis-flex-algo)# metric-type delay
Router(config-isis-flex-algo)# advertise-definition
Router(config-isis-flex-algo)# affinity include-any RED
```

2. Configure the loopback interface and associate the IPv4 and IPv6 addresses with the flexible algorithm.

```
Router(config)# interface Loopback51
Router(config-if)# passive
Router(config-if)# ipv4 address 192.0.2.51 255.255.255.255 algorithm 151
```

```
Router(config-if)# ipv6 address 2001:DB8:2:2::51/128 algorithm 151
Router(config-if)# commit
```

3. Optional: Associate secondary interface addresses with the same flexible algorithm.

```
Router(config)# interface HundredGigE0/1/0/0.301
Router(config-if)# ipv4 address 198.51.100.1 255.255.255.0 secondary algorithm
 151
Router(config-if)# ipv6 address 2001:DB8:200:3:2:1::1/112 algorithm 151
Router(config-if)# encapsulation dot1q 301
Router(config-if)# commit
```

4. Verify the IS-IS flexible algorithm configuration.

```
Router# show run router isis
router isis 1
 is-type level-2-only
 net 49.0000.0000.0000.0012.00
 affinity-map RED bit-position 4
 flex-algo 151
  data-plane ip
  metric-type delay
  advertise-definition
  affinity include-any RED
 !
!
```

5. Verify the loopback address association.

```
Router# show running interface Loopback51
interface Loopback51
 ipv4 address 192.0.2.51 255.255.255.255 algorithm 151
 ipv6 address 2001:DB8:2:2::51/128 algorithm 151
!
```

6. Verify the secondary interface address association.

```
Router# show running interface HundredGigE0/1/0/0.301
interface HundredGigE0/1/0/0.301
 ipv4 address 198.51.100.1 255.255.255.0 secondary algorithm 151
 ipv6 address 2001:DB8:200:3:2:1::1/112 algorithm 151
 encapsulation dot1q 301
!
```

The router advertises flexible algorithm 151 for the IP data plane and associates the configured IPv4 and IPv6 interface addresses with that flexible algorithm.

Verify IP flexible algorithm

Teaches you how to verify flexible algorithm definitions, prefix advertisements, data-plane participation, topology entries, routes, and fast-reroute paths by using IS-IS show commands.

Use this task to verify that IP flexible algorithm is enabled, advertised, and installed correctly in the IS-IS topology and forwarding plane.

Use the **show isis flex-algo** command to verify the data plane that is used with the flexible algorithm configuration. You can also verify prefix advertisements, topology entries, route installation, and backup-path information.

Before you begin, configure the flexible algorithm and associate the required IPv4 and IPv6 addresses with that algorithm.

1. Verify the flexible algorithm definition and confirm that the IP data plane is enabled.

```
Router# show isis flex-algo 151
```

Confirm that the output shows `Data Plane Segment Routing: No` and `Data Plane IP: Yes`.

```
Router# show isis flex-algo 151

IS-IS 1 Flex-Algo Database

Flex-Algo 151:

  Level-2:
    Definition Priority: 128
    Definition Source: Router.00, (Local)
    Definition Equal to Local: Yes
    Definition Metric Type: Delay
    Definition Flex-Algo Prefix Metric: No
    Exclude Any Affinity Bit Positions:
    Include Any Affinity Bit Positions:  4
    Include All Affinity Bit Positions:
    Exclude SRLGs:
    Disabled: No

  Local Priority: 128
  FRR Disabled: No
  Microloop Avoidance Disabled: No

  Data Plane Segment Routing: No
  Data Plane IP: Yes
```

2. Verify the IP flexible algorithm prefix advertisements in the IS-IS database.

```
Router# show isis database detail verbose RouterR2
```

Confirm that the output shows IPv4 and IPv6 algorithm prefixes for algorithm 151.

```
Router# show isis database detail verbose RouterR2

IS-IS 1 (Level-2) Link State Database
LSPID                 LSP Seq Num  LSP Checksum  LSP Holdtime/Rcvd  ATT/P/OL
RouterR2.00-00        0x0000004c   0xd97e        773  /1200         0/0/0
  Area Address:   49.0000
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 192.0.2.31/32 D:0 Metric: 0
 Algorithm: 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
    Source Router ID: 192.0.2.2
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 192.0.2.51/32 D:0 Metric: 0
 Algorithm: 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
    Source Router ID: 192.0.2.2
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 198.51.100.0/24 D:0 Metric:
 10 Algorithm: 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 203.0.113.0/24 D:0 Metric:
10 Algorithm: 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv6 Algo Prefix: MT (IPv6 Unicast) 2001:DB8:2:2::31/128 D:0 Metric: 0
Algorithm: 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv6 Algo Prefix: MT (IPv6 Unicast) 2001:DB8:2:2::51/128 D:0 Metric: 0
```

```
Algorithm: 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  NLPID:          0xcc
  NLPID:          0x8e
  IP Address:     192.0.2.2
  Router ID:      192.0.2.2
  Metric: 0          IP-Extended 192.0.2.2/32
    Prefix-SID Index: 2, Algorithm:0, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix-SID Index: 102, Algorithm:131, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix Attribute Flags: X:0 R:0 N:1 E:0 A:0
    Source Router ID: 192.0.2.2
  Metric: 0          MT (IPv6 Unicast) IPv6 2001:DB8:2:2::2/128
    Prefix-SID Index: 12, Algorithm:0, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix-SID Index: 112, Algorithm:131, R:0 N:1 P:0 E:0 V:0 L:0
```

3. Verify that flexible algorithm 151 is present only in the IP data plane topology.

```
Router# show isis topology flex-algo 151 data-plane segment-routing
Router# show isis topology flex-algo 151 data-plane ip
```

Confirm that the segment-routing data plane does not contain valid paths for flexible algorithm 151 and that the IP data plane shows the expected next hops and interfaces.

```
Router# show isis topology flex-algo 151 data-plane segment-routing

IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id          Metric    Next-Hop           Interface       SNPA
RouterR1             --
RouterR2             **
RouterR3             **
RouterR4             **
RouterR5             **
RouterR6             **
```

```
Router# show isis topology flex-algo 151 data-plane ip

IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id          Metric    Next-Hop           Interface       SNPA
RouterR1             --
RouterR2             32        RouterR3         Te0/0/0/14.2    *PtoP*
RouterR3             20        RouterR3         BE1             *PtoP*
RouterR4             30        RouterR3         BE1             *PtoP*
RouterR5             **        RouterR5         Te0/0/0/14.2    *PtoP*
RouterR6             **        RouterR5         Te0/0/0/14.2    *PtoP*
```

4. Verify the IPv4 and IPv6 routes and fast-reroute information for the flexible algorithm prefixes.

```
Router# show isis route flex-algo 151 192.0.2.51/32 detail
Router# show isis fast-reroute flex-algo 151 192.0.2.51/32 detail
Router# show isis ipv6 route flex-algo 151 2001:DB8:2:2::51/128 detail
Router# show isis ipv6 fast-reroute flex-algo 151 2001:DB8:2:2::51/128 detail
```

Confirm that the routes are installed and that the output shows the expected primary path and LFA backup path for both IPv4 and IPv6 prefixes.

```
Router# show isis route flex-algo 151 192.0.2.51/32 detail

L2 192.0.2.51/32 [32/115] Label: None, medium priority
   Installed for 03:04:11
```

```
     via 203.0.113.2, TenGigE0/0/0/14.2, RouterR4, SRGB Base: 17000, Weight:
 0
     src RouterR2.00-00, 192.0.2.2
```

```
Router# show isis fast-reroute flex-algo 151 192.0.2.51/32 detail

L2 192.0.2.51/32 [32/115] Label: None, medium priority
   Installed for 03:04:25
     via 203.0.113.2, TenGigE0/0/0/14.2, RouterR4, SRGB Base: 17000, Weight:
 0
      Backup path: LFA, via 198.51.100.2, Bundle-Ether1, RouterR2, SRGB Base:
 17000, Weight: 0, Metric: 40
       P: No, TM: 40, LC: No, NP: Yes, D: Yes, SRLG: No
     src RouterR2.00-00, 192.0.2.2
```

```
Router# show isis ipv6 route flex-algo 151 2001:DB8:2:2::51/128 detail

L2 2001:DB8:2:2::51/128 [32/115] Label: None, medium priority
   Installed for 00:17:33
     via fe80::28a:96ff:fee7:f400, TenGigE0/0/0/14.2, RouterR4, SRGB Base:
17000, Weight: 0
     src RouterR2.00-00, 2001:DB8:2:2::2
```

```
Router# show isis ipv6 fast-reroute flex-algo 151 2001:DB8:2:2::51/128 detail

L2 2001:DB8:2:2::51/128 [32/115] Label: None, medium priority
   Installed for 00:17:45
     via fe80::28a:96ff:fee7:f400, TenGigE0/0/0/14.2, RouterR4, SRGB Base:
17000, Weight: 0
      Backup path: LFA, via fe80::2bc:60ff:fe04:74dc, Bundle-Ether1, RouterR2,
 SRGB Base: 17000, Weight: 0, Metric: 40
       P: No, TM: 40, LC: No, NP: Yes, D: Yes, SRLG: No
     src RouterR2.00-00, 2001:DB8:2:2::2
```

Protecting IP flexible algorithm prefixes

Explains how IP flexible algorithm traffic can be protected by Segment Routing-based TI-LFA and microloop avoidance when both IP and Segment Routing data-planes are enabled with matching participating routers.

Protecting Flexible Algorithm IP prefixes is a protection method that

- uses Segment Routing to provide TI-LFA and microloop avoidance protection for IP flexible algorithm prefixes
- requires Segment Routing and IP flexible algorithm data planes to be enabled for the same flexible algorithm in an IS-IS area, and
- requires the same set of reachable routers in the IS-IS area to participate in both data planes for that flexible algorithm.

IP flexible algorithm does not support TI-LFA and microloop avoidance independently. To protect IP flexible algorithm prefixes in the IP network, you must enable Segment Routing in the network.

TI-LFA protection or microloop avoidance for IP flexible algorithm traffic requires these conditions:

- Both the Segment Routing and IP flexible algorithm data planes must be enabled for the same flexible algorithm in an IS-IS area.
- The same set of reachable routers in the IS-IS area must participate in both the Segment Routing and IP data planes for that flexible algorithm.

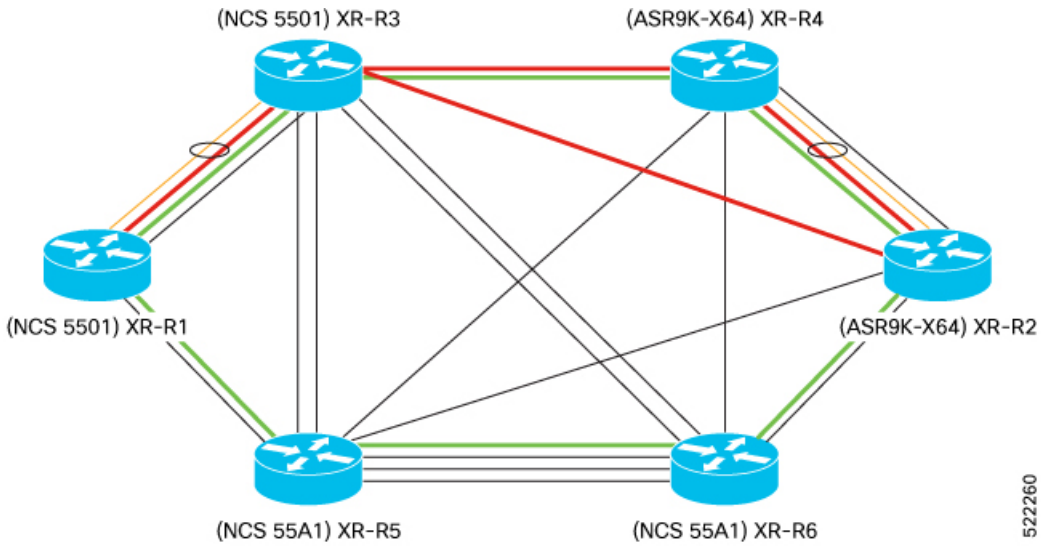
Configure flexible algorithm protection

Shows how to configure and verify protection for flexible algorithm traffic by enabling both Segment Routing and IP data-planes, associating prefixes with algorithms, and validating TI-LFA protection on matching topologies.

Use this task to enable protection for IP flexible algorithm traffic by configuring Segment Routing flexible algorithm and IP flexible algorithm in the same IS-IS area.

This example uses a six-node topology from R1 to R6. In this topology, node R2 advertises the winning flexible algorithm definition.

Figure 7:



Flexible algorithm 131 is enabled for both the Segment Routing and IP data planes. Flexible algorithm 151 is enabled for the IP data plane.

Before you begin, configure IS-IS in the area, define the required affinity maps, and determine the flexible algorithm numbers, metric types, and interface addresses that you want to use.

1. Configure flexible algorithm 131 for both the Segment Routing and IP data planes, and configure flexible algorithm 151 for the IP data plane.

```
Router(config)# router isis 1
Router(config-isis)# is-type level-2-only
Router(config-isis)# net 49.0000.0000.0000.0012.00
Router(config-isis)# affinity-map GREEN bit-position 4
Router(config-isis)# flex-algo 131
Router(config-isis-flex-algo)# data-plane segment-routing ip
Router(config-isis-flex-algo)# metric-type delay
Router(config-isis-flex-algo)# advertise-definition
Router(config-isis-flex-algo)# affinity include-any GREEN
Router(config-isis-flex-algo)# exit
Router(config-isis)# flex-algo 151
Router(config-isis-flex-algo)# data-plane ip
Router(config-isis-flex-algo)# metric-type delay
Router(config-isis-flex-algo)# advertise-definition
Router(config-isis-flex-algo)# affinity include-any RED
```

2. Enable Segment Routing protection for IPv4 and IPv6 unicast address families.

```
Router(config-isis)# address-family ipv4 unicast
Router(config-isis-af)# metric-style wide
```

```
Router(config-isis-af)# segment-routing mpls sr-prefer
Router(config-isis-af)# segment-routing mpls unlabeled protection route-policy
 ip_fa
Router(config-isis-af)# exit
Router(config-isis)# address-family ipv6 unicast
Router(config-isis-af)# metric-style wide
Router(config-isis-af)# segment-routing mpls sr-prefer
Router(config-isis-af)# segment-routing mpls unlabeled protection route-policy
 ip_fa
```

3. Configure prefix SIDs for flexible algorithm 131 on the loopback interface.

```
Router(config)# interface Loopback0
Router(config-if)# passive
Router(config-if)# address-family ipv4 unicast
Router(config-if-af)# prefix-sid index 2
Router(config-if-af)# prefix-sid algorithm 131 index 102
Router(config-if-af)# exit
Router(config-if)# address-family ipv6 unicast
Router(config-if-af)# prefix-sid index 12
Router(config-if-af)# prefix-sid algorithm 131 index 112
```

4. Associate loopback prefixes with flexible algorithms 131 and 151.

```
Router(config)# interface Loopback31
Router(config-if)# ipv4 address 192.0.2.31 255.255.255.255 algorithm 131
Router(config-if)# ipv6 address 2001:DB8:2:2::31/128 algorithm 131
Router(config)# interface Loopback51
Router(config-if)# ipv4 address 192.0.2.51 255.255.255.255 algorithm 151
Router(config-if)# ipv6 address 2001:DB8:2:2::51/128 algorithm 151
```

5. Associate the physical interface prefixes with flexible algorithms 131 and 151.

```
Router(config)# interface HundredGigE0/1/0/0.301
Router(config-if)# ipv4 address 198.51.100.1 255.255.255.0 algorithm 131
Router(config-if)# ipv4 address 203.0.113.1 255.255.255.0 secondary algorithm
 151
Router(config-if)# ipv6 address 2001:DB8:200:3:1:1::1/112 algorithm 131
Router(config-if)# ipv6 address 2001:DB8:200:3:2:1::1/112 algorithm 151
Router(config-if)# encapsulation dot1q 301
Router(config-if)# commit
```

The router is configured to protect IP flexible algorithm traffic by using flexible algorithm 131 in both the Segment Routing and IP data planes, while flexible algorithm 151 remains enabled for the IP data plane.

6. Use these steps to verify the flexible algorithm protection configuration.
   a) Verify flexible algorithm 131 in both data planes.

```
Router# show isis flex-algo 131

IS-IS 1 Flex-Algo Database

Flex-Algo 131:

  Level-2:
    Definition Priority: 128
    Definition Source: Router.00, (Local)
    Definition Equal to Local: Yes
```

```
    Definition Metric Type: Delay
    Definition Flex-Algo Prefix Metric: No
    Exclude Any Affinity Bit Positions:
    Include Any Affinity Bit Positions:  4
    Include All Affinity Bit Positions:
    Exclude SRLGs:
    Disabled: No

  Local Priority: 128
  FRR Disabled: No
  Microloop Avoidance Disabled: No

  Data Plane Segment Routing: Yes
  Data Plane IP: Yes
```

b) Verify flexible algorithm 151 in the IP data plane.

```
Router# show isis flex-algo 151

IS-IS 1 Flex-Algo Database

Flex-Algo 151:

  Level-2:
    Definition Priority: 128
    Definition Source: Router.00, (Local)
    Definition Equal to Local: Yes
    Definition Metric Type: Delay
    Definition Flex-Algo Prefix Metric: No
    Exclude Any Affinity Bit Positions:
    Include Any Affinity Bit Positions:  4
    Include All Affinity Bit Positions:
    Exclude SRLGs:
    Disabled: No

  Local Priority: 128
  FRR Disabled: No
  Microloop Avoidance Disabled: No

  Data Plane Segment Routing: No
  Data Plane IP: Yes
```

c) Verify interface participation.

```
Router# show isis interface Loopback31

Loopback31                  Enabled
  Adjacency Formation:      Disabled (Passive in IS-IS cfg)
  Prefix Advertisement:     Enabled
 !
  IPv4 Address Family:      Enabled
    Protocol State:         Up
    Forwarding Address(es): Unknown (Intf passive in IS-IS cfg)
    Global Prefix(es):      192.0.2.31/32 (131)
  IPv6 Address Family:      Enabled
    Protocol State:         Up
    Forwarding Address(es): Unknown (Intf passive in IS-IS cfg)
    Global Prefix(es):      2001:DB8:2:2::31/128 (131)
!
```

d) Verify flexible algorithm advertisements in the IS-IS database.

```
Router# show isis database detail verbose RouterR2
IS-IS 1 (Level-2) Link State Database
LSPID                 LSP Seq Num  LSP Checksum  LSP Holdtime/Rcvd  ATT/P/OL
RouterR2.00-00          0x0000004c   0xd97e        773  /1200         0/0/0

  Area Address:   49.0000
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 2.2.2.31/32 D:0 Metric:
0 Algorithm: 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
    Source Router ID: 2.2.2.2
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 2.2.2.51/32 D:0 Metric:
0 Algorithm: 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
    Source Router ID: 2.2.2.2
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 200.3.1.0/24 D:0 Metric:
 10 Algorithm: 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 200.3.2.0/24 D:0 Metric:
 10 Algorithm: 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
 IPv6 Algo Prefix: MT (IPv6 Unicast) 2:2:2::31/128 D:0 Metric: 0 Algorithm:
 131
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
 IPv6 Algo Prefix: MT (IPv6 Unicast) 2:2:2::51/128 D:0 Metric: 0 Algorithm:
 151
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  NLPID:          0xcc
  NLPID:          0x8e
  IP Address:     2.2.2.2
  Router ID:      2.2.2.2
  Metric: 0          IP-Extended 2.2.2.2/32
    Prefix-SID Index: 2, Algorithm:0, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix-SID Index: 102, Algorithm:131, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix Attribute Flags: X:0 R:0 N:1 E:0 A:0
    Source Router ID: 2.2.2.2
  Metric: 0          MT (IPv6 Unicast) IPv6 2:2:2::2/128
    Prefix-SID Index: 12, Algorithm:0, R:0 N:1 P:0 E:0 V:0 L:0
    Prefix-SID Index: 112, Algorithm:131, R:0 N:1 P:0 E:0 V:0 L:0
....
....
Router Cap:     2.2.2.2 D:0 S:0
    Segment Routing: I:1 V:1, SRGB Base: 17000 Range: 7000
    SR Local Block: Base: 15000 Range: 1000
    Node Maximum SID Depth:
      Label Imposition: 10
    SR Algorithm:
      Algorithm: 0
      Algorithm: 131
    IP Algorithm:
      Algorithm: 131
      Algorithm: 151
    Flex-Algo Definition:
      Algorithm: 131 Metric-Type: 1 Alg-type: 0 Priority: 128
      Flex-Algo Include-Any Ext Admin Group:
        0x00000010
    Flex-Algo Definition:
      Algorithm: 151 Metric-Type: 1 Alg-type: 0 Priority: 128
      Flex-Algo Include-Any Ext Admin Group:
        0x00000010
```

e) Verify topology congruence for flexible algorithm 131.

```
Router# show isis topology flex-algo 131 data-plane segment-routing

IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id          Metric    Next-Hop           Interface       SNPA

RouterR1             --
RouterR2             32        RouterR5            Te0/0/0/14.2    *PtoP*

RouterR3             20        RouterR3            BE1             *PtoP*

RouterR4             30        RouterR3            BE1             *PtoP*

RouterR5             12        RouterR5            Te0/0/0/14.2    *PtoP*

RouterR6             22        RouterR5            Te0/0/0/14.2    *PtoP*
```

```
Router# show isis topology flex-algo 131 data-plane ip

IS-IS 1 paths to IPv4 Unicast (Level-2) routers
System Id          Metric    Next-Hop           Interface       SNPA

RouterR1             --
RouterR2             32        RouterR5            Te0/0/0/14.2    *PtoP*

RouterR3             20        RouterR3            BE1             *PtoP*

RouterR4             30        RouterR3            BE1             *PtoP*

RouterR5             12        RouterR5            Te0/0/0/14.2    *PtoP*

RouterR6             22        RouterR5            Te0/0/0/14.2    *PtoP*
```

f) Verify Segment Routing and IP flexible algorithm route protection.

```
Router# show isis route flex-algo 131 192.0.2.101/32 detail

L2 192.0.2.101/32 [31/115] Label: 17101, medium priority
   Installed Nov 15 13:46:40.902 for 01:46:13
     via 32.1.24.2, TenGigE0/1/0/3/6, Label: 17101, RouterR4, SRGB Base:
17000, Weight: 0
     src RouterR1.00-00, 192.0.2.101, prefix-SID index 101, R:0 N:1 P:0
E:0 V:0 L:0, Alg:131
```

```
Router# show isis fast-reroute flex-algo 131 192.0.2.101/32 detail
L2 192.0.2.101/32 [31/115] Label: 17101, medium priority
   Installed Nov 15 13:46:40.902 for 01:46:19
     via 32.1.24.2, TenGigE0/1/0/3/6, Label: 17101, RouterR4, SRGB Base:
17000, Weight: 0
      Backup path: TI-LFA (link), via 32.1.26.2, TenGigE0/1/0/3/5.2 tb5-r6,
 SRGB Base: 17000, Weight: 0, Metric: 120
        P node: RouterR6.00 [6.6.6.6], Label: ImpNull
        Q node: RouterR5.00 [5.5.5.5], Label: 25009
        Prefix label: 17101
        Backup-src: RouterR1.00
       P: No, TM: 120, LC: No, NP: No, D: No, SRLG: No
```

```
     src RouterR1.00-00, 192.0.2.101, prefix-SID index 101, R:0 N:1 P:0
E:0 V:0 L:0, Alg:131
```

```
Router# show isis route flex-algo 131 192.0.2.31/32 detail
L2 192.0.2.31/32 [31/115] Label: None, medium priority
   Installed Nov 15 13:46:34.923 for 01:46:42
     via 32.1.24.2, TenGigE0/1/0/3/6, tb5-r4, SRGB Base: 17000, Weight: 0
```

```
Router# show isis fast-reroute flex-algo 131 192.0.2.31/32 detail
L2 192.0.2.31/32 [31/115] Label: None, medium priority
   Installed Nov 15 13:46:34.923 for 01:46:54
     via 32.1.24.2, TenGigE0/1/0/3/6, tb5-r4, SRGB Base: 17000, Weight: 0
      Backup path: TI-LFA (link), via 32.1.26.2, TenGigE0/1/0/3/5.2 tb5-r6,
 SRGB Base: 17000, Weight: 0, Metric: 120
        P node: tb5-r6.00 [6.6.6.6], Label: ImpNull
        Q node: tb5-r5.00 [5.5.5.5], Label: 25009
        Prefix label: None
        Backup-src: RouterR1.00
       P: No, TM: 120, LC: No, NP: No, D: No, SRLG: No
```

Flexible-algorithm redistribution in IP networks

Explains how redistributed prefixes can be assigned to a specific flexible algorithm instead of algorithm 0, allowing constrained path computation and algorithm-based route matching.

Flexible-algorithm redistribution in IP networks is a routing capability that

- redistributes prefixes into a specific flexible algorithm in the range `128` to `255`
- lets link-state IGPs compute paths by using flexible algorithm constraints for redistributed prefixes, and
- allows you to set an algorithm on prefixes and match prefixes based on that algorithm.

In earlier releases, redistributed flexible algorithm prefixes from another domain were placed in the base algorithm, which is algorithm 0. As a result, routes could not be redistributed into a specific flexible algorithm from another domain.

Starting in Release 7.9.1, routers can redistribute prefixes into a specified flexible algorithm. This capability preserves flexible algorithm intent for redistributed prefixes and applies constraint-based path computation to those prefixes in the receiving domain.

This feature supports these functions:

- Setting an algorithm in redistributed prefixes
- Matching redistributed prefixes based on the algorithm

Restrictions for flexible-algorithm redistribution

- The route policy accepts algorithm numbers from 0 through 255.
- IS-IS handles only algorithm 0 and algorithms 128 through 255.
- Algorithm numbers 1 through 127 in IS-IS are misconfigured algorithms and are treated as algorithm 0. If you redistribute flexible algorithm prefixes in the range 1 through 127 from another domain, all prefixes are redistributed into the base algorithm, which is algorithm 0.

Set an algorithm

Explains how to use route policy set algorithm to assign redistributed prefixes to a specific flexible algorithm and verify that the algorithm is attached in the IS-IS database.

Use this task to set a flexible algorithm for prefixes that are redistributed into IS-IS by using the **set algorithm** statement in route policy language.

The **set algorithm** command is available in the route-policy mechanism. You can use this command to set the flexible algorithm for prefixes that are redistributed into IS-IS protocols. The prefix advertisement after route redistribution can be altered through the **set algorithm** statement in the routing policy.

Before you begin, identify the IPv4 and IPv6 prefixes that you want to redistribute and the flexible algorithm that you want to assign to those prefixes.

1. Create a prefix set for the prefixes that you want to redistribute into flexible algorithm 128.

```
Router(config)# prefix-set PFX_ALGO128
Router(config-pfx)# 44.44.44.128/32,
Router(config-pfx)# 44:44:44::128/128
Router(config-pfx)# end-set
```

2. Define a route policy that sets flexible algorithm 128 for the prefix set.

```
Router(config)# route-policy BGP_TO_ISIS
Router(config-rpl)# if destination in PFX_ALGO128 then
Router(config-rpl-if)# set tag 200
Router(config-rpl-if)# set algorithm 128
Router(config-rpl-if)# pass
Router(config-rpl-if)# else
Router(config-rpl-else)# drop
Router(config-rpl-else)# endif
Router(config-rpl)# end-policy
Router(config)# commit
```

3. Apply the route policy while redistributing BGP routes into IS-IS for IPv4 and IPv6 unicast address families.

```
Router(config)# router isis 100
Router(config-isis)# address-family ipv4 unicast
Router(config-isis-af)# redistribute bgp 100 route-policy BGP_TO_ISIS
metric-type rib-metric-as-external
Router(config-isis-af)# exit
Router(config-isis)# address-family ipv6 unicast
Router(config-isis-af)# redistribute bgp 100 route-policy BGP_TO_ISIS
metric-type rib-metric-as-external
Router(config-isis-af)# commit
Router(config-isis-af)# end
```

The prefixes in prefix set `PFX_ALGO128` are redistributed into IS-IS with flexible algorithm 128 set by the route policy.

4. Verify redistributed prefixes in the IS-IS database.

Confirm that the RPL is applied and the redistributed prefixes have flexible algorithm 128 attached.

```
Router# show isis instance 100 database R4.00-01 verbose | begin
44.44.44.128/32
Tue Dec 13 09:12:41.395 UTC
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 44.44.44.128/32 D:0 Metric:
 2 Algorithm: 128
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
```

```
     algo 128 due to RPL with set algo.
    Admin. Tag: 200
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 44.44.44.128/32 D:0 Metric:
 10 Algorithm: 129
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 44.44.44.128/32 D:0 Metric:
 10 Algorithm: 130
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 44.44.44.128/32 D:0 Metric:
 10 Algorithm: 131
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  IPv4 Algo Prefix: MT (Standard (IPv4 Unicast)) 44.44.44.128/32 D:0 Metric:
 10 Algorithm: 132
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  Metric: 0          MT (IPv6 Unicast) IPv6-External 6:6:6:6::6/128
    Admin. Tag: 200
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  Metric: 10         MT (IPv6 Unicast) IPv6-External 7:7:7:7::7/128
    Admin. Tag: 200
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv6 Algo Prefix: MT (IPv6 Unicast) 44:44:44::132/128 D:0 Metric: 0
Algorithm: 132
    Prefix Attribute Flags: X:0 R:0 N:0 E:0 A:0
  IPv6 Algo Prefix: MT (IPv6 Unicast) 66:66:66::128/128 D:0 Metric: 0
Algorithm: 128
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  IPv6 Algo Prefix: MT (IPv6 Unicast) 44:44:44::128/128 D:0 Metric: 2
Algorithm: 128
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
  IPv6 Algo Prefix: MT (IPv6 Unicast) 44:44:44::129/128 D:0 Metric: 10
Algorithm: 129
    Prefix Attribute Flags: X:1 R:0 N:0 E:0 A:0
    Admin. Tag: 200
```

Match an algorithm

Explains how to use route policy match conditions to filter redistributed routes by flexible algorithm number and verify the selected prefixes in IS-IS and BGP outputs.

Use this task to configure route policy language to match algorithm numbers and filter prefixes during route redistribution.

You can configure match algorithms in route policy language and use them to filter prefixes based on algorithm numbers during redistribution. This mechanism acts as a filter based on the algorithm number in the prefixes while redistributing routes.

You can use **match algorithm** as a conditional expression to select or filter prefixes that match a specific algorithm during route redistribution.

Before you begin, identify the algorithm numbers that you want to allow during redistribution and the routing protocol into which you want to redistribute the filtered routes.

1. Define a route policy that matches flexible algorithm 128 and algorithm 0 prefixes.

```
Router(config)# route-policy ISIS_TO_BGP
Router(config-rpl)# if algorithm is 128 or algorithm is 0 then
Router(config-rpl-if)# pass
```

```
Router(config-rpl-if)# else
Router(config-rpl-else)# drop
Router(config-rpl-else)# endif
Router(config-rpl)# end-policy
Router(config)# commit
```

2. Apply the route policy while redistributing IS-IS routes into BGP for IPv4 and IPv6 unicast address families.

```
Router(config)# router bgp 100
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# redistribute isis 100 route-policy ISIS_TO_BGP
Router(config-bgp-af)# exit
Router(config-bgp)# address-family ipv6 unicast
Router(config-bgp-af)# redistribute isis 100 route-policy ISIS_TO_BGP
Router(config-bgp-af)# commit
```

The route policy filters redistributed IS-IS routes so that only prefixes with flexible algorithm 128 or algorithm 0 are redistributed into BGP.

3. Verify routes in flexible algorithm 128.

```
Router# show isis route flex-algo 128
Tue Dec 13 09:34:09.502 UTC
IS-IS 100 IPv4 Unicast routes Flex-Algo 128
Codes: L1 - level 1, L2 - level 2, ia - interarea (leaked into level 1)
       df - level 1 default (closest attached router), su - summary null
       C - connected, S - static, R - RIP, B - BGP, O - OSPF
       E - EIGRP, A - access/subscriber, M - mobile, a - application
       i - IS-IS (redistributed from another instance)
Maximum parallel path count: 8
L2 11.11.11.128/32 [20/115]
     via 2.4.0.2, HundredGigE0/0/0/1, R2, SRGB Base: 16000, Weight: 0
L2 22.22.22.128/32 [10/115]
     via 2.4.0.2, HundredGigE0/0/0/1, R2, SRGB Base: 16000, Weight: 0
L2 33.33.33.128/32 [30/115]
     via 2.4.0.2, HundredGigE0/0/0/1, R2, SRGB Base: 16000, Weight: 0
C  44.44.44.128/32
     is directly connected, Loopback129
L2 55.55.55.128/32 [40/115]
     via 2.4.0.2, HundredGigE0/0/0/1, R2, SRGB Base: 16000, Weight: 0
```

4. Verify route details for a flexible algorithm 128 prefix.

```
Router# show route 11.11.11.128/32 detail
Tue Dec 13 09:35:16.681 UTC
Routing entry for 11.11.11.128/32
  Known via "isis 100", distance 115, metric 4 (algo 128), type level-2
  Installed Dec 13 07:56:46.972 for 01:38:29
  Routing Descriptor Blocks
    4.5.0.5, from 1.1.1.1, via HundredGigE0/0/0/2, Backup (Local-LFA)
      Route metric is 54
      Label: None
      Tunnel ID: None
      Binding Label: None
      Extended communities count: 0
      Path id:65        Path ref count:1
      NHID:0x3(Ref:29)
    2.4.0.2, from 1.1.1.1, via HundredGigE0/0/0/1, Protected
      Route metric is 4
      Label: None
```

```
      Tunnel ID: None
      Binding Label: None
      Extended communities count: 0
      Path id:1         Path ref count:0
      NHID:0x2(Ref:32)
      Backup path id:65
  Route version is 0x2 (2)
  No local label
  IP Precedence: Not Set
  QoS Group ID: Not Set
  Flow-tag: Not Set
  Fwd-class: Not Set
  Route Priority: RIB_PRIORITY_NON_RECURSIVE_MEDIUM (7) SVD Type
RIB_SVD_TYPE_LOCAL
  Download Priority 1, Download Version 170
  No advertising protos.
```

5. Verify BGP advertised prefixes after route policy filtering.

```
Router# show bgp ipv4 unicast advertised summary
Tue Dec 13 09:37:53.596 UTC
Network            Next Hop        From            Advertised to
1.1.1.1/32         4.6.0.4         Local           4.6.0.6
1.2.0.0/24         4.6.0.4         Local           4.6.0.6
1.3.0.0/24         4.6.0.4         Local           4.6.0.6
2.2.2.2/32         4.6.0.4         Local           4.6.0.6
2.4.0.0/24         4.6.0.4         Local           4.6.0.6
3.3.3.3/32         4.6.0.4         Local           4.6.0.6
3.5.0.0/24         4.6.0.4         Local           4.6.0.6
4.4.4.4/32         4.6.0.4         Local           4.6.0.6
4.5.0.0/24         4.6.0.4         Local           4.6.0.6
4.6.0.0/24         4.6.0.4         Local           4.6.0.6
5.5.5.5/32         4.6.0.4         Local           4.6.0.6
5.6.0.0/24         4.6.0.4         Local           4.6.0.6
11.11.11.128/32    4.6.0.4         Local           4.6.0.6
22.22.22.128/32    4.6.0.4         Local           4.6.0.6
33.33.33.128/32    4.6.0.4         Local           4.6.0.6
44.44.44.128/32    4.6.0.4         Local           4.6.0.6
55.55.55.128/32    4.6.0.4         Local           4.6.0.6
Processed 17 prefixes, 17 paths
```