# BGP Configuration Guide for Cisco NCS 560 Series Routers, IOS XR Release 6.6.x

**First Published:** 2019-05-30

**Last Modified:** 2019-12-13

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

# CONTENTS

CHAPTER 1

# Implementing BGP

Border Gateway Protocol (BGP) is an Exterior Gateway Protocol (EGP) that allows you to create loop-free interdomain routing between autonomous systems. An *autonomous system* is a set of routers under a single technical administration. Routers in an autonomous system can use multiple Interior Gateway Protocols (IGPs) to exchange routing information inside the autonomous system and an EGP to route packets outside the autonomous system.

This module provides conceptual and configuration information on BGP.



# BGP Functional Overview

***Table 1: Feature History Table***

| Feature Name | Release Information | Feature Description |
|---|---|---|
| BGP Fallback Feature for LAG Bundles | Release 7.5.1 | This feature is now supported on routers that have Cisco NC57 line cards installed and operate in native and compatibiltiy mode.<br>This feature enables recreate the essence of OSPF Cost Fallback feature for LAG bundles for directly connected BGP sessions from a PE to a core router. |

BGP uses TCP as its transport protocol. Two BGP routers form a TCP connection between one another (peer routers) and exchange messages to open and confirm the connection parameters.

BGP routers exchange network reachability information. This information is mainly an indication of the full paths (BGP autonomous system numbers) that a route should take to reach the destination network. This information helps construct a graph that shows which autonomous systems are loop free and where routing policies can be applied to enforce restrictions on routing behavior.

Any two routers forming a TCP connection to exchange BGP routing information are called peers or neighbors. BGP peers initially exchange their full BGP routing tables. After this exchange, incremental updates are sent as the routing table changes. BGP keeps a version number of the BGP table, which is the same for all of its BGP peers. The version number changes whenever BGP updates the table due to routing information changes. Keepalive packets are sent to ensure that the connection is alive between the BGP peers and notification packets are sent in response to error or special conditions.

> **Note** VPNv4 address family is supported effective from Cisco IOS XR Release 6.1.31. However, VPNv6 and VPN routing and forwarding (VRF) address families will be supported in a future release.

# Enable BGP Routing

Perform this task to enable BGP routing and establish a BGP routing process. Configuring BGP neighbors is included as part of enabling BGP routing.

> **Note**
> - At least one neighbor and at least one address family must be configured to enable BGP routing. At least one neighbor with both a remote AS and an address family must be configured globally using the **address family** and **remote as** commands.
> - When one BGP session has both IPv4 unicast and IPv4 labeled-unicast AFI/SAF, then the routing behavior is nondeterministic. Therefore, the prefixes may not be correctly advertised. Incorrect prefix advertisement results in reachability issues. In order to avoid such reachability issues, you must explicitly configure a route policy to advertise prefixes either through IPv4 unicast or through IPv4 labeled-unicast address families.

### Before you begin

BGP must be able to obtain a router identifier (for example, a configured loopback address). At least, one address family must be configured in the BGP router configuration and the same address family must also be configured under the neighbor.

> **Note** If the neighbor is configured as an external BGP (eBGP) peer, you must configure an inbound and outbound route policy on the neighbor using the **route-policy** command.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# route-policy drop-as-1234
  RP/0/RP0/CPU0:router(config-rpl)# if as-path passes-through '1234' then
  RP/0/RP0/CPU0:router(config-rpl)# apply check-communities
```

```
RP/0/RP0/CPU0:router(config-rpl)# else
RP/0/RP0/CPU0:router(config-rpl)# pass
RP/0/RP0/CPU0:router(config-rpl)# endif
```

(Optional) Creates a route policy and enters route policy configuration mode, where you can define the route policy.

**Step 3** **end-policy**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

(Optional) Ends the definition of a route policy and exits route policy configuration mode.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

**Step 5** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 6** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 7** **bgp router-id** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp router-id 192.168.70.24
```

Configures the local router with a specified router ID.

**Step 8** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 9** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# exit
```

Exits the current configuration mode.

**Step 10** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 11** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 12** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 13** **route-policy** *route-policy-name* { **in** | **out** }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy drop-as-1234 in
```

(Optional) Applies the specified policy to inbound IPv4 unicast routes.

**Step 14** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Enabling BGP: Example

The following shows how to enable BGP.

```
prefix-set static
   2020::/64,
   2012::/64,
   10.10.0.0/16,
   10.2.0.0/24
end-set

route-policy pass-all
  pass
end-policy
route-policy set_next_hop_agg_v4
  set next-hop 10.0.0.1
end-policy

route-policy set_next_hop_static_v4
  if (destination in static) then
    set next-hop 10.1.0.1
  else
    drop
  endif
end-policy

route-policy set_next_hop_agg_v6
  set next-hop 2003::121
end-policy

route-policy set_next_hop_static_v6
  if (destination in static) then
     set next-hop 2011::121
  else
     drop
  endif
end-policy

router bgp 65000
  bgp fast-external-fallover disable
  bgp confederation peers
    65001
    65002
  bgp confederation identifier 1
  bgp router-id 1.1.1.1
  address-family ipv4 unicast
    aggregate-address 10.2.0.0/24 route-policy set_next_hop_agg_v4
    aggregate-address 10.3.0.0/24
    redistribute static route-policy set_next_hop_static_v4
  address-family ipv6 unicast
    aggregate-address 2012::/64 route-policy set_next_hop_agg_v6
    aggregate-address 2013::/64
    redistribute static route-policy set_next_hop_static_v6
    neighbor 10.0.101.60
    remote-as 65000
    address-family ipv4 unicast
    neighbor 10.0.101.61
    remote-as 65000
    address-family ipv4 unicast
    neighbor 10.0.101.62
    remote-as 3
    address-family ipv4 unicast
      route-policy pass-all in
      route-policy pass-all out
    neighbor 10.0.101.64
    remote-as 5
    update-source Loopback0
```

```
address-family ipv4 unicast
  route-policy pass-all in
  route-policy pass-all out
```

# Adjust BGP Timers

BGP uses certain timers to control periodic activities, such as the sending of keepalive messages and the interval after which a neighbor is assumed to be down if no messages are received from the neighbor during the interval. The values set using the **timers bgp** command in router configuration mode can be overridden on particular neighbors using the **timers** command in the neighbor configuration mode.

Perform this task to set the timers for BGP neighbors.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 123
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **timers bgp** *keepalive hold-time*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# timers bgp 30 90
```

Sets a default keepalive time and a default hold time for all neighbors.

**Step 4** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 5** **timers** *keepalive hold-time*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# timers 60 220
```

(Optional) Sets the keepalive timer and the hold-time timer for the BGP neighbor.

**Step 6** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Change BGP Default Local Preference Value

Perform this task to set the default local preference value for BGP paths.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **bgp default local-preference** *value*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp default local-preference 200
```

Sets the default local preference value from the default of 100, making it either a more preferable path (over 100) or less preferable path (under 100).

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.

- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Configure MED Metric for BGP

Perform this task to set the multi exit discriminator (MED) to advertise to peers for routes that do not already have a metric set (routes that were received with no MED attribute).

**Procedure**

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **default-metric** *value*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# default-metric
```

Sets the default metric, which is used to set the MED to advertise to peers for routes that do not already have a metric set (routes that were received with no MED attribute).

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Configure BGP Weights

A weight is a number that you can assign to a path so that you can control the best-path selection process. If you have particular neighbors that you want to prefer for most of your traffic, you can use the **weight** command to assign a higher weight to all routes learned from that neighbor. Perform this task to assign a weight to routes received from a neighbor.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 5** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 6** **weight** *weight-value*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# weight 41150
```

Assigns a weight to all routes learned through the neighbor.

**Step 7** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### What to do next

You the **clear bgp** command for the newly configured weight to take effect.

# Tune BGP Best-Path Calculation

BGP routers typically receive multiple paths to the same destination. The BGP best-path algorithm determines the best path to install in the IP routing table and to use for forwarding traffic. The BGP best-path comprises of three steps:

- Step 1—Compare two paths to determine which is better.
- Step 2—Iterate over all paths and determines which order to compare the paths to select the overall best path.
- Step 3—Determine whether the old and new best paths differ enough so that the new best path should be used.

**Note** The order of comparison determined by Step 2 is important because the comparison operation is not transitive; that is, if three paths, A, B, and C exist, such that when A and B are compared, A is better, and when B and C are compared, B is better, it is not necessarily the case that when A and C are compared, A is better. This nontransitivity arises because the multi exit discriminator (MED) is compared only among paths from the same neighboring autonomous system (AS) and not among all paths. BGP Best Path Algorithm, on page 133 provides additional conceptual details.

Perform this task to change the default BGP best-path calculation behavior.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 126
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **bgp bestpath med missing-as-worst**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp bestpath med missing-as-worst
```

Directs the BGP software to consider a missing MED attribute in a path as having a value of infinity, making this path the least desirable path.

**Step 4** **bgp bestpath med always**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp bestpath med always
```

Configures the BGP speaker in the specified autonomous system to compare MEDs among all the paths for the prefix, regardless of the autonomous system from which the paths are received.

**Step 5** **bgp bestpath med confed**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp bestpath med confed
```

Enables BGP software to compare MED values for paths learned from confederation peers.

**Step 6** **bgp bestpath as-path ignore**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp bestpath as-path ignore
```

Configures the BGP software to ignore the autonomous system length when performing best-path selection.

**Step 7** **bgp bestpath compare-routerid**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp bestpath compare-routerid
```

Configure the BGP speaker in the autonomous system to compare the router IDs of similar paths.

**Step 8** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Set BGP Administrative Distance

An administrative distance is a rating of the trustworthiness of a routing information source. In general, the higher the value, the lower the trust rating. Normally, a route can be learned through more than one protocol. Administrative distance is used to discriminate between routes learned from more than one protocol. The route with the lowest administrative distance is installed in the IP routing table. By default, BGP uses the administrative distances shown in here:

*Table 2: BGP Default Administrative Distances*

| Distance | Default Value | Function |
|---|---|---|
| External | 20 | Applied to routes learned from eBGP. |
| Internal | 200 | Applied to routes learned from iBGP. |
| Local | 200 | Applied to routes originated by the router. |

**Note** Distance does not influence the BGP path selection algorithm, but it does influence whether BGP-learned routes are installed in the IP routing table.

Perform this task to specify the use of administrative distances that can be used to prefer one class of route over another.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **distance bgp** *external-distance internal-distance local-distance*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# distance bgp 20 20 200
```

Sets the external, internal, and local administrative distances to prefer one class of routes over another. The higher the value, the lower the trust rating.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Indicate BGP Back-door Routes

In most cases, when a route is learned through eBGP, it is installed in the IP routing table because of its distance. Sometimes, however, two ASs have an IGP-learned back-door route and an eBGP-learned route. Their policy might be to use the IGP-learned path as the preferred path and to use the eBGP-learned path when the IGP path is down.

Perform this task to set the administrative distance on an external Border Gateway Protocol (eBGP) route to that of a locally sourced BGP route, causing it to be less preferred than an Interior Gateway Protocol (IGP) route.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **network** { *ip-address* / *prefix-length* | *ip-address mask* } **backdoor**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# network 172.20.0.0/16
```

Configures the local router to originate and advertise the specified network.

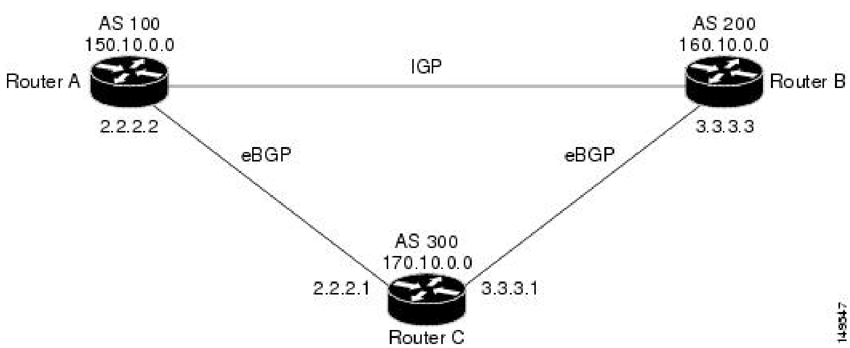
**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Back Door: Example




Here, Routers A and C and Routers B and C are running eBGP. Routers A and B are running an IGP (such as Routing Information Protocol [RIP], Interior Gateway Routing Protocol [IGRP], Enhanced IGRP, or Open Shortest Path First [OSPF]). The default distances for RIP, IGRP, Enhanced IGRP, and OSPF are 120, 100, 90, and 110, respectively. All these distances are higher than the default distance of eBGP, which is 20. Usually, the route with the lowest distance is preferred.

Router A receives updates about 160.10.0.0 from two routing protocols: eBGP and IGP. Because the default distance for eBGP is lower than the default distance of the IGP, Router A chooses the eBGP-learned route from Router C. If you want Router A to learn about 160.10.0.0 from Router B (IGP), establish a BGP back door. See .

In the following example, a network back-door is configured:

```
RP/0/RP0/CPU0:router(config)# router bgp 100
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-bgp-af)# network 160.10.0.0/16 backdoor
```

Router A treats the eBGP-learned route as local and installs it in the IP routing table with a distance of 200. The network is also learned through Enhanced IGRP (with a distance of 90), so the Enhanced IGRP route is successfully installed in the IP routing table and is used to forward traffic. If the Enhanced IGRP-learned route goes down, the eBGP-learned route is installed in the IP routing table and is used to forward traffic.

Although BGP treats network 160.10.0.0 as a local entry, it does not advertise network 160.10.0.0 as it normally would advertise a local entry.

# Configure Aggregate Addresses

Perform this task to create aggregate entries in a BGP routing table.

**Procedure**

---

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **aggregate-address** *address/mask-length* [ **as-set** ] [ **as-confed-set** ] [ **summary-only** ] [ **route-policy** *route-policy-name* ]

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# aggregate-address 10.0.0.0/8 as-set
```

Creates an aggregate address. The path advertised for this route is an autonomous system set consisting of all elements contained in all paths that are being summarized.

- The **as-set** keyword generates autonomous system set path information and community information from contributing paths.
- The **as-confed-set** keyword generates autonomous system confederation set path information from contributing paths.
- The **summary-only** keyword filters all more specific routes from updates.
- The **route-policy** *route-policy-name* keyword and argument specify the route policy used to set the attributes of the aggregate route.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Understanding BGP MD5 Authentication

BGP provides a mechanism, known as Message Digest 5 (MD5) authentication, for authenticating a TCP segment between two BGP peers by using a clear text or encrypted password.

MD5 authentication is configured at the BGP neighbor level. BGP peers using MD5 authentication are configured with the same password. If the password authentication fails, then the packets are not transmitted along the segment.

## Configuring BGP MD5 Authentication

You can use the configuration in this section to configure BGP MD5 authentication between two BGP peers.

**Note** The configuration for MD5 authentication is identical on both peers.

### Configuration

Use the following configuration to configure BGP MD5:

```
RP/0/RP0/CPU0:router(config)# router bgp 50
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-bgp-af)# exit
```

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.1.1.1
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 51
RP/0/RP0/CPU0:router(config-bgp-nbr)# password encrypted a1b2c3
RP/0/RP0/CPU0:router(config-bgp-nbr)# commit
```

### Running Configuration

Validate the configuration.

```
RP/0/RP0/CPU0:router# show running-config
...
!
router bgp 50
address-family ipv4 unicast
!
neighbor 10.1.1.1
remote-as 51
password encrypted a1b2c3
!
!
```

# Hiding the Local AS Number for BGP Networks

Changing the autonomous system number is necessary when two separate BGP networks are combined under a single autonomous system. The neighbor **local-as** command is used to configure BGP peers to support two local autonomous system numbers to maintain peering between two separate BGP networks.

However, when the neighbor **local-as** command is configured on a BGP peer, the local AS number is automatically prepended to all routes that are learned from eBGP peers by default. This behavior, however, makes changing the autonomous system number for a service provider or large BGP network difficult, because the routes with the prepended AS number are rejected by internal BGP (iBGP) peers that belong to the same AS.

Hiding the local AS number by using the **no-prepend** command simplifies the process of changing the autonomous system number in a Border Gateway Protocol (BGP) network. Without this feature, internal BGP (iBGP) peers reject external routes from peers with a local AS number in the as-path attribute to prevent routing loops. Hiding the local AS number allows you to transparently change the autonomous system number for the entire BGP network and ensure that routes can be propagated throughout the autonomous system, while the AS number transition is incomplete.

## Configuring BGP to Hide the Local AS Number

Hiding the local AS number for eBGP peers by using the **no-prepend** command can be used to transparently change the AS number of a BGP network, and ensure that routes are propagated throughout the AS during the transition. Because the local AS number is not prepended to these routes, external routes are not rejected by internal peers during the transition from one AS number to another.

This section describes the configuration and verification of the feature.

**Note** BGP prepends the autonomous system number from each BGP network that a route traverses. This behavior is designed to maintain network reachability information and to prevent routing loops from occurring. Configuring the **no-prepend** command incorrectly could create routing loops. So, the configuration of this command should only be attempted by an experienced network operator.

#### Configuration

Use the following configuration to hide the local AS number for eBGP peers.

```
Router# config
Router(config)# router bgp 100
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# network 172.20.1.1 255.255.240.0
Router(config-bgp-af)# neighbor 172.20.1.1
Router(config-bgp-af)# remote-as 150
Router(config-bgp-af)# local-as 300 no-prepend
Router(config-bgp-af)# commit
```

#### Running Configuration

```
RP/0/RP0/CPU0:router# show running-configuration
...
!
router bgp 100
 address-family ipv4 unicast
 network 10.1.1.1 255.255.0.0
 neighbor 10.1.1.1 remote-as 100
 neighbor 10.1.1.1 local-as 300 no-prepend
!
```

#### Verification

Use the following command to verify your configuration.

```
RP/0/RP0/CPU0:router# show ip bgp neighbors
BGP neighbor is 10.1.1.1, remote AS 100, local AS 300 no-prepend, external link
BGP version 4, remote router ID 10.1.1.1
BGP state = Established, up for 00:00:49
Last read 00:00:49, hold time is 180, keepalive interval is 60 seconds
Neighbor capabilities:
Route refresh: advertised and received(new)
Address family IPv4 Unicast: advertised and received
IPv4 MPLS Label capability:
Received 10 messages, 1 notifications, 0 in queue
Sent 10 messages, 0 notifications, 0 in queue
Default minimum time between advertisement runs is 30 seconds
```

# Autonomous System Number Formats in BGP

Autonomous system numbers (ASNs) are globally unique identifiers used to identify autonomous systems (ASs) and enable ASs to exchange exterior routing information between neighboring ASs. A unique ASN is allocated to each AS for use in BGP routing. ASNs are encoded as 2-byte numbers and 4-byte numbers in BGP.

```
RP/0/RP0/CPU0:router(config)# as-format [asdot | asplain]
RP/0/RP0/CPU0:router(config)# as-format asdot
```

## 2-byte Autonomous System Number Format

The 2-byte ASNs are represented in asplain notation. The 2-byte range is 1 to 65535.

## 4-byte Autonomous System Number Format

To prepare for the eventual exhaustion of 2-byte Autonomous System Numbers (ASNs), BGP has the capability to support 4-byte ASNs. The 4-byte ASNs are represented both in asplain and asdot notations.

The byte range for 4-byte ASNs in asplain notation is 1-4294967295. The AS is represented as a 4-byte decimal number. The 4-byte ASN asplain representation is defined in draft-ietf-idr-as-representation-01.txt.

For 4-byte ASNs in asdot format, the 4-byte range is 1.0 to 65535.65535 and the format is:

*high-order-16-bit-value-in-decimal . low-order-16-bit-value-in-decimal*

The BGP 4-byte ASN capability is used to propagate 4-byte-based AS path information across BGP speakers that do not support 4-byte AS numbers. See draft-ietf-idr-as4bytes-12.txt for information on increasing the size of an ASN from 2 bytes to 4 bytes. AS is represented as a 4-byte decimal number

## as-format Command

The **as-format** command configures the ASN notation to asdot. The default value, if the **as-format** command is not configured, is asplain.

## BGP Multi-Instance and Multi-AS

Multi-AS BGP enables configuring each instance of a multi-instance BGP with a different AS number. Multi-Instance and Multi-AS BGP provides these capabilities:

- Mechanism to consolidate the services provided by multiple routers using a common routing infrastructure into a single IOS-XR router.
- Mechanism to achieve AF isolation by configuring the different AFs in different BGP instances.
- Means to achieve higher session scale by distributing the overall peering sessions between multiple instances.
- Mechanism to achieve higher prefix scale (especially on a RR) by having different instances carrying different BGP tables.
- Improved BGP convergence under certain scenarios.
- All BGP functionalities including NSR are supported for all the instances.
- The load and commit router-level operations can be performed on previously verified or applied configurations.

### Restrictions

- The router supports maximum of 4 BGP instances.
- Each BGP instance needs a unique router-id.
- Only one Address Family can be configured under each BGP instance (VPNv4, VPNv6 and RT-Constrain can be configured under multiple BGP instances).
- IPv4/IPv6 Unicast should be within the same BGP instance in which IPv4/IPv6 Labeled-Unicast is configured.
- IPv4/IPv6 Multicast should be within the same BGP instance in which IPv4/IPv6 Unicast is configured.

- All configuration changes for a single BGP instance can be committed together. However, configuration changes for multiple instances cannot be committed together.
- Cisco recommends that BGP update-source should be unique in the default VRF over all instances while peering with the same remote router.

## Configure Multiple BGP Instances for a Specific Autonomous System

Perform this task to configure multiple BGP instances for a specific autonomous system. All configuration changes for a single BGP instance can be committed together. However, configuration changes for multiple instances cannot be committed together.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number* [**instance** *instance name*]

**Example:**

```
RP/0/RSP0/CPU0:router(config)# router bgp 100 instance inst1
```

Enters BGP configuration mode for the user specified BGP instance.

**Step 3** **bgp** **router-id***ip-address*

**Example:**

```
RP/0/RSP0/CPU0:router(config-bgp)# bgp router-id 10.0.0.0
```

Configures a fixed router ID for the BGP-speaking router (BGP instance).

**Note** You must manually configure unique router ID for each BGP instance.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# BGP Routing Domain Confederation

One way to reduce the iBGP mesh is to divide an autonomous system into multiple sub-autonomous systems and group them into a single confederation. To the outside world, the confederation looks like a single autonomous system. Each autonomous system is fully meshed within itself and has a few connections to other autonomous systems in the same confederation. Although the peers in different autonomous systems have eBGP sessions, they exchange routing information as if they were iBGP peers. Specifically, the next hop, MED, and local preference information is preserved. This feature allows you to retain a single IGP for all of the autonomous systems.

## Configure Routing Domain Confederation for BGP

Perform this task to configure the routing domain confederation for BGP. This includes specifying a confederation identifier and autonomous systems that belong to the confederation.

Configuring a routing domain confederation reduces the internal BGP (iBGP) mesh by dividing an autonomous system into multiple autonomous systems and grouping them into a single confederation. Each autonomous system is fully meshed within itself and has a few connections to another autonomous system in the same confederation. The confederation maintains the next hop and local preference information, and that allows you to retain a single Interior Gateway Protocol (IGP) for all autonomous systems. To the outside world, the confederation looks like a single autonomous system.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **bgp confederation identifier** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation identifier 5
```

Specifies a BGP confederation identifier.

**Step 4** **bgp confederation peers** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1091
  RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1092
```

```
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1093
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1094
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1095
RP/0/RP0/CPU0:router(config-bgp)# bgp confederation peers 1096
```

Specifies that the BGP autonomous systems belong to a specified BGP confederation identifier. You can associate multiple AS numbers to the same confederation identifier, as shown in the example.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### BGP Confederation: Example

The following is a sample configuration that shows several peers in a confederation. The confederation consists of three internal autonomous systems with autonomous system numbers 6001, 6002, and 6003. To the BGP speakers outside the confederation, the confederation looks like a normal autonomous system with autonomous system number 666 (specified using the **bgp confederation identifier** command).

In a BGP speaker in autonomous system 6001, the **bgp confederation peers** command marks the peers from autonomous systems 6002 and 6003 as special eBGP peers. Hence, peers 171.16 .232.55 and 171.16 .232.56 get the local preference, next hop, and MED unmodified in the updates. The router at 171 .19 .69.1 is a normal eBGP speaker, and the updates received by it from this peer are just like a normal eBGP update from a peer in autonomous system 666.

```
router bgp 6001
 bgp confederation identifier 666
 bgp confederation peers
  6002
  6003
   exit
 address-family ipv4 unicast
  neighbor 171.16.232.55
  remote-as 6002
   exit
 address-family ipv4 unicast
  neighbor 171.16.232.56
  remote-as 6003
   exit
 address-family ipv4 unicast
  neighbor 171.19.69.1
  remote-as 777
```

In a BGP speaker in autonomous system 6002, the peers from autonomous systems 6001 and 6003 are configured as special eBGP peers. Peer 171 .17 .70.1 is a normal iBGP peer, and peer 199.99.99.2 is a normal eBGP peer from autonomous system 700.

```
router bgp 6002
 bgp confederation identifier 666
 bgp confederation peers
  6001
  6003
   exit
 address-family ipv4 unicast
  neighbor 171.17.70.1
   remote-as 6002
   exit
 address-family ipv4 unicast
  neighbor 171.19.232.57
   remote-as 6001
   exit
 address-family ipv4 unicast
  neighbor 171.19.232.56
   remote-as 6003
   exit
 address-family ipv4 unicast
  neighbor 171.19.99.2
   remote-as 700
   exit
 address-family ipv4 unicast
  route-policy pass-all in
  route-policy pass-all out
```

In a BGP speaker in autonomous system 6003, the peers from autonomous systems 6001 and 6002 are configured as special eBGP peers. Peer 192 .168 .200.200 is a normal eBGP peer from autonomous system 701.

```
router bgp 6003
 bgp confederation identifier 666
 bgp confederation peers
  6001
  6002
   exit
 address-family ipv4 unicast
  neighbor 171.19.232.57
   remote-as 6001
   exit
 address-family ipv4 unicast
  neighbor 171.19.232.55
   remote-as 6002
   exit
 address-family ipv4 unicast
  neighbor 192.168.200.200
   remote-as 701
   exit
 address-family ipv4 unicast
  route-policy pass-all in
  route-policy pass-all out
```

The following is a part of the configuration from the BGP speaker 192 .168 .200.205 from autonomous system 701 in the same example. Neighbor 171.16 .232.56 is configured as a normal eBGP speaker from autonomous system 666. The internal division of the autonomous system into multiple autonomous systems is not known to the peers external to the confederation.

```
router bgp 701
 address-family ipv4 unicast
  neighbor 172.16.232.56
   remote-as 666
   exit
 address-family ipv4 unicast
  route-policy pass-all in
  route-policy pass-all out
   exit
 address-family ipv4 unicast
  neighbor 192.168.200.205
   remote-as 701
```

# BGP Additional Paths

The Border Gateway Protocol (BGP) Additional Paths feature modifies the BGP protocol machinery for a BGP speaker to be able to send multiple paths for a prefix. This gives 'path diversity' in the network. The add path enables BGP prefix independent convergence (PIC) at the edge routers.

BGP add path enables add path advertisement in an iBGP network and advertises the following types of paths for a prefix:

- Backup paths—to enable fast convergence and connectivity restoration.
- Group-best paths—to resolve route oscillation.
- All paths—to emulate an iBGP full-mesh.

## Configure BGP Additional Paths

Perform these tasks to configure BGP Additional Paths capability:

**Procedure**

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router (config)#route-policy add_path_policy
```

Defines the route policy and enters route-policy configuration mode.

**Step 3** **if** *conditional-expression* **then** *action-statement* **else**

**Example:**

```
RP/0/RP0/CPU0:router (config-rpl)#if community matches-any (*) then
    set path-selection all advertise
  else
```

Decides the actions and dispositions for the given route.

**Step 4** **pass endif**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl-else)#pass
RP/0/RP0/CPU0:router(config-rpl-else)#endif
```

Passes the route for processing and ends the if statement.

**Step 5** **end-policy**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)#end-policy
```

Ends the route policy definition of the route policy and exits route-policy configuration mode.

**Step 6** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)#router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 7** **address-family** {**ipv4** {**unicast** } | **ipv6** {**unicast** | **l2vpn vpls-vpws** | **vpnv4 unicast** | **vpnv6 unicast** }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)#address-family ipv4 unicast
```

Specifies the address family and enters address family configuration submode.

**Step 8** **additional-paths receive**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)#additional-paths receive
```

Configures receive capability of multiple paths for a prefix to the capable peers.

**Step 9** **additional-paths send**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)#additional-paths send
```

Configures send capability of multiple paths for a prefix to the capable peers .

**Step 10** **additional-paths selection route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)#additional-paths selection route-policy add_path_policy
```

Configures additional paths selection capability for a prefix.

**Step 11** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# BGP Maximum Prefix

The maximum-prefix feature imposes a maximum limit on the number of prefixes that are received from a neighbor for a given address family. Whenever the number of prefixes received exceeds the maximum number configured, the BGP session is terminated, which is the default behavior, after sending a cease notification to the neighbor. The session is down until a manual clear is performed by the user. The session can be resumed by using the **clear bgp** command. It is possible to configure a period after which the session can be automatically brought up by using the **maximum-prefix** command with the **restart** keyword. The maximum prefix limit can be configured by the user. Default limits are used if the user does not configure the maximum number of prefixes for the address family.

**Discard Extra Paths**

An option to discard extra paths is added to the maximum-prefix configuration. Configuring the discard extra paths option drops all excess prefixes received from the neighbor when the prefixes exceed the configured maximum value. This drop does not, however, result in session flap.

The benefits of discard extra paths option are:

- Limits the memory footstamp of BGP.
- Stops the flapping of the peer if the paths exceed the set limit.

When the discard extra paths configuration is removed, BGP sends a route-refresh message to the neighbor if it supports the refresh capability; otherwise the session is flapped.

On the same lines, the following describes the actions when the maximum prefix value is changed:

- If the maximum value alone is changed, a route-refresh message is sourced, if applicable.
- If the new maximum value is greater than the current prefix count state, the new prefix states are saved.
- If the new maximum value is less than the current prefix count state, then some existing prefixes are deleted to match the new configured state value.

There is currently no way to control which prefixes are deleted.

## Configure Discard Extra Paths

The discard extra paths option in the maximum-prefix configuration allows you to drop all excess prefixes received from the neighbor when the prefixes exceed the configured maximum value. This drop does not, however, result in session flap.

The benefits of discard extra paths option are:

- Limits the memory footstamp of BGP.
- Stops the flapping of the peer if the paths exceed the set limit.

When the discard extra paths configuration is removed, BGP sends a route-refresh message to the neighbor if it supports the refresh capability; otherwise the session is flapped.

**Note**

- When the router drops prefixes, it is inconsistent with the rest of the network, resulting in possible routing loops.
- If prefixes are dropped, the standby and active BGP sessions may drop different prefixes. Consequently, an NSR switchover results in inconsistent BGP tables.
- The discard extra paths configuration cannot co-exist with the *soft reconfig* configuration.

Perform this task to configure BGP maximum-prefix discard extra paths.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters XR Config mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 10
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.0.0.1
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

**Step 5** **maximum-prefix** *maximum* **discard-extra-paths**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# maximum-prefix 1000 discard-extra-paths
```

Configures a limit to the number of prefixes allowed.

Configures discard extra paths to discard extra paths when the maximum prefix limit is exceeded.

**Step 6** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

---

### Example

The following example shows how to configure discard extra paths feature for the IPv4 address family:

```
RP/0//CPU0:router# configure
RP/0//CPU0:router(config)# router bgp 10
RP/0//CPU0:router(config-bgp)# neighbor 10.0.0.1
RP/0//CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
RP/0//CPU0:router(config-bgp-nbr-af)# maximum-prefix 1000 discard-extra-paths
RP/0//CPU0:router(config-bgp-vrf-af)# commit
```

The following screen output shows details about the discard extra paths option:

```
RP/0//CPU0:ios# show bgp neighbor 10.0.0.1

BGP neighbor is 10.0.0.1
Remote AS 10, local AS 10, internal link
Remote router ID 0.0.0.0
BGP state = Idle (No best local address found)
Last read 00:00:00, Last read before reset 00:00:00
Hold time is 180, keepalive interval is 60 seconds
Configured hold time: 180, keepalive: 60, min acceptable hold time: 3
Last write 00:00:00, attempted 0, written 0
Second last write 00:00:00, attempted 0, written 0
Last write before reset 00:00:00, attempted 0, written 0
Second last write before reset 00:00:00, attempted 0, written 0
Last write pulse rcvd not set last full not set pulse count 0
Last write pulse rcvd before reset 00:00:00
Socket not armed for io, not armed for read, not armed for write
Last write thread event before reset 00:00:00, second last 00:00:00
Last KA expiry before reset 00:00:00, second last 00:00:00
Last KA error before reset 00:00:00, KA not sent 00:00:00
Last KA start before reset 00:00:00, second last 00:00:00
Precedence: internet
Multi-protocol capability not received
Received 0 messages, 0 notifications, 0 in queue
Sent 0 messages, 0 notifications, 0 in queue
Minimum time between advertisement runs is 0 secs

For Address Family: IPv4 Unicast
```

```
BGP neighbor version 0
Update group: 0.1 Filter-group: 0.0 No Refresh request being processed
Route refresh request: received 0, sent 0
0 accepted prefixes, 0 are bestpaths
Cumulative no. of prefixes denied: 0.
Prefix advertised 0, suppressed 0, withdrawn 0
Maximum prefixes allowed 10 (discard-extra-paths) <<<<<<<<<<<<<<<<<<<<<
Threshold for warning message 75%, restart interval 0 min
AIGP is enabled
An EoR was not received during read-only mode
Last ack version 1, Last synced ack version 0
Outstanding version objects: current 0, max 0
Additional-paths operation: None
Send Multicast Attributes

Connections established 0; dropped 0
Local host: 0.0.0.0, Local port: 0, IF Handle: 0x00000000
Foreign host: 10.0.0.1, Foreign port: 0
Last reset 00:00:00
```

# BGP Best-External Path

*Table 3: Feature History Table*

| Feature Name | Release Information | Feature Description |
|---|---|---|
| BGP Best-External for VPN Address Family Identifier and Subaddress Family Identifier | Release 7.5.1 | This feature is now supported on routers that have Cisco NC57 line cards installed and operate in native and compatibiltiy mode.<br><br>The feature advertises a best external route to its internal peers as a backup route. The backup route is stored in the RIB and Cisco Express Forwarding. If the primary path fails, the BGP PIC functionality enables the best external path to take over, enabling faster restoration of connectivity. |

The best–external path functionality supports advertisement of the best–external path to the iBGP and Route Reflector peers when a locally selected bestpath is from an internal peer. BGP selects one best path and one backup path to every destination. By default, selects one best path . Additionally, BGP selects another bestpath from among the remaining external paths for a prefix. Only a single path is chosen as the best–external path and is sent to other PEs as the backup path. BGP calculates the best–external path only when the best path is an iBGP path. If the best path is an eBGP path, then best–external path calculation is not required.

The procedure to determine the best–external path is as follows:

1. Determine the best path from the entire set of paths available for a prefix.
2. Eliminate the current best path.
3. Eliminate all the internal paths for the prefix.
4. From the remaining paths, eliminate all the paths that have the same next hop as that of the current best path.
5. Rerun the best path algorithm on the remaining set of paths to determine the best–external path.

BGP considers the external and confederations BGP paths for a prefix to calculate the best–external path. BGP advertises the best path and the best–external path as follows:

- On the primary PE—advertises the best path for a prefix to both its internal and external peers
- On the backup PE—advertises the best path selected for a prefix to the external peers and advertises the best–external path selected for that prefix to the internal peers

## Configure Best-External Path Advertisement

Perform the following tasks to advertise the best–external path to the iBGP and route-reflector peers:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** Do one of the following

- **address-family { vpnv4 unicast | vpnv6 unicast }**
- **vrf***vrf-name***{ipv4 unicast|ipv6 unicast}**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family vpnv4 unicast
```

Specifies the address family or VRF address family and enters the address family or VRF address family configuration submode.

**Step 4** **advertise best-external**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# advertise best-external
```

Advertise the best–external path to the iBGP and route-reflector peers.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# BGP Local Label Retention

When a primary PE-CE link fails, BGP withdraws the route corresponding to the primary path along with its local label and programs the backup path in the Routing Information Base (RIB) and the Forwarding Information Base (FIB), by default.

However, until all the internal peers of the primary PE reconverge to use the backup path as the new bestpath, the traffic continues to be forwarded to the primary PE with the local label that was allocated for the primary path. Hence the previously allocated local label for the primary path must be retained on the primary PE for some configurable time after the reconvergence. BGP Local Label Retention feature enables the retention of the local label for a specified period. If no time is specified, the local lable is retained for a default value of five minutes.

## Retain Allocated Local Label for Primary Path

Perform the following tasks to retain the previously allocated local label for the primary path on the primary PE for some configurable time after reconvergence:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family { vpnv4 unicast | vpnv6 unicast }**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family vpnv4 unicast
```

Specifies the address family and enters the address family configuration submode.

**Step 4** **retain local-label** *minutes*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# retain local-label 10
```

Retains the previously allocated local label for the primary path on the primary PE for 10 minutes after reconvergence.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Allocated Local Label Retention: Example

The following example shows how to retain the previously allocated local label for the primary path on the primary PE for 10 minutes after reconvergence:

```
router bgp 100
address-family l2vpn vpnv4 unicast
  retain local-label 10
end
```

# BGP Labeled Unicast Multiple Label Stack Overview

BGP Labeled Unicast Multiple Label Stack feature enables the user to make the XR router receive and advertise BGP LU updates with a stack of one or more labels associated with the encoded prefix.

This feature provides the ability for a controller to push a multiple label stack through BGP labeled unicast session onto the headend.

### Prerequisites

BGP Labelled unicast address-family needs to be supported.

### Restrictions

Due to hardware limitations, only a maximum of three label stacks is supported; from Release 6.6.1, a maximum of five labels are supported

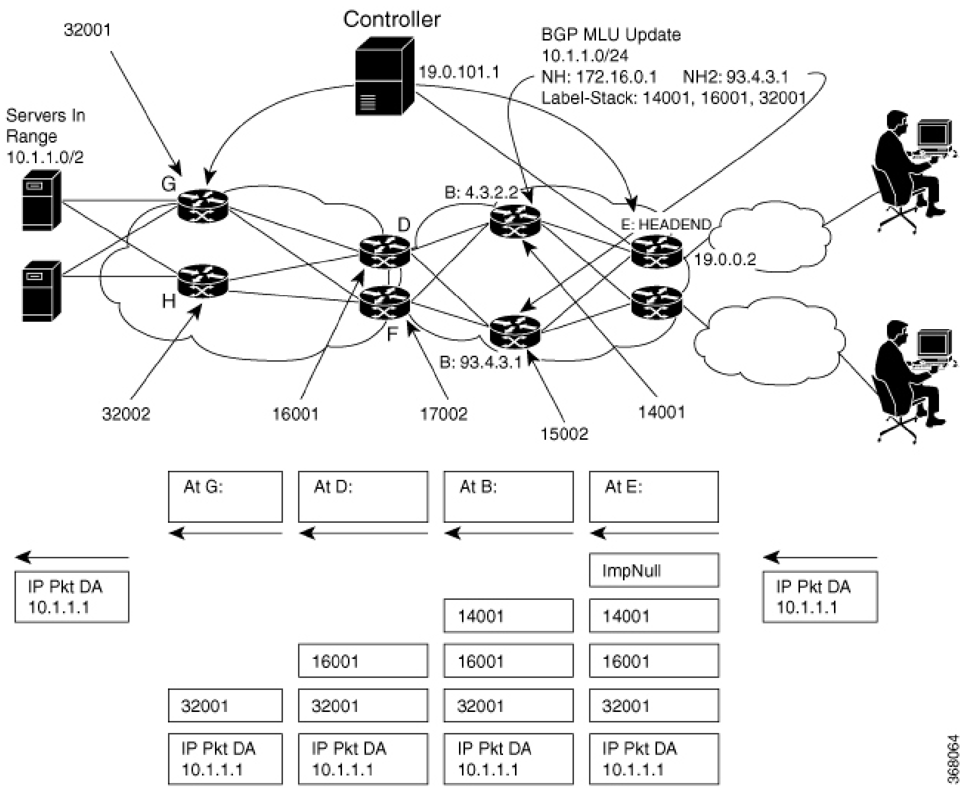
### Topology

The following section illustrates the topology for the BGP Labeled Unicast Multiple Label Stack feature.

Based on the multi-label stack pushed by the controller on to the head end E, the traffic is steered through the network. In this topology, as the controller is pushing the label stack 14001, 16001, and 32001 with NH

172.6.0.1, traffic is steered through the nodes B, D, and G sequentially. If the controller needs to change the traffic path to nodes C, F, and G sequentially, it pushes the label stack 15002, 17002, and 32001 with NH of 93.4.3.1.

*Figure 1: BGP Labeled Unicast Multiple Label Stack Topology*




## Configuration

This section describes how you can configure the BGP Labeled Unicast Multiple Label Stack feature.

Configure the **nexthop mpls forwarding ibgp** command in BGP configuration mode. Configure the BGP labeled unicast session with Nexthop 10.3.2.2 so the "ImpNULL" label is pushed as the first label into the multiple-label stack.

```
Router# configure
Router(config)# router bgp 100
Router(config-bgp)# neighbor 10.0.1.101
Router(config-bgp)# nexthop mpls forwarding ibgp
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# allocate-label all
Router(config-bgp-af)# exit
Router(config-bgp)# neighbor 10.3.2.2
Router(config-bgp-nbr)# remote-as 100
Router(config-bgp-nbr)# address-family ipv4 labeled-unicast
Router(config-bgp)# exit
Router(config-bgp)# neighbor-group group 1
Router(config-bgp-nbrgrp)# neighbor-group group 1
Router(config-bgp-nbrgrp)# remote-as 65535
Router(config-bgp-nbrgrp)# address-family ipv4 labeled-unicast
Router(config-bgp-nbrgrp-af)# route-policy pass in
```

```
Router(config-bgp-nbrgrp-af)# route-policy pass out
Router(config-bgp-nbrgrp-af)# enforce-multiple-labels
Router(config-bgp-nbrgrp-af)# exit
Router(config-bgp-nbrgrp)# exit
Router(config-bgp)# neighbor 10.0.1.101
Router(config-bgp-nbr)# use neighbor-group ipv4lu_ng1
Router(config-bgp-nbr)# exit
Router(config-bgp)# exit
Router(config-bgp)# neighbor 10.0.1.101
Router(config-bgp-nbr)# remote-as 65535
Router(config-bgp-nbr)# address-family ipv4 labeled-unicast
Router(config-bgp-nbr-af)# route-policy pass in
Router(config-bgp-nbr-af)# route-policy pass out
Router(config-bgp-nbr-af)# route-reflector-client
Router(config-bgp-nbr-af)# enforce-multiple-labels
```

### Running Configuration

```
router bgp 100
bgp router-id 10.0.1.101
nexthop mpls forwarding ibgp
address-family ipv4 unicast
  allocate-label all
!
 neighbor 10.3.2.2
  remote-as 100
  address-family ipv4 labeled-unicast
!
neighbor-group ipv4lu_ng1
  remote-as 100
  address-family ipv4 labeled-unicast
   route-policy pass in
   route-policy pass out
   enforce-multiple-labels

neighbor 10.0.1.101
  use neighbor-group ipv4lu_ng1
  !
!
neighbor 10.0.1.101
  remote-as 100
  address-family ipv4 labeled-unicast
   route-policy pass out
   route-policy pass in
   route-reflector-client
   enforce-multiple-labels
!
```

## Verification

The show outputs given in the following section display the details of configuration of the BGP LU Multiple Label Stack feature, and the status of their configuration.

```
/* Verify the multiple label stack. */
Router# show bgp ipv4 labeled-unicast 10.1.1.1/32

...
```

```
10.3.2.2 from 10.0.1.101

      Received Label 14001 16001 32001

      Origin incomplete, metric 0, localpref 94, valid, internal, best, group-best

      Received Path ID 0, Local Path ID 0, version 42

      Community: 258:259 260:261 262:263 264:265

      Large Community: 1:2:3 5:6:7
...

/* Verify if the multiple label stack is enabled.*/
Router# show bgp neighbor 10.0.1.101

...

 For Address Family: IPv4 Labeled-unicast

  BGP neighbor version 177675

  Update group: 0.8 Filter-group: 0.4  No Refresh request being processed

  Route-Reflector Client

  Send Multicast Attributes

 Multiple label stack: Enabled


/* Verify that the multiple label stack is enabled. */
Router# show bgp ipv4 labeled-unicast update-group 0.8

Update group for IPv4 Labeled-unicast, index 0.8:

  Attributes:

    Neighbor sessions are IPv4

    Outbound policy: ibgp-rpl1

    Internal

    Common admin

    First neighbor AS: 100

    Send communities

    Send GSHUT community if originated

    Send extended communities

    Route Reflector Client

    4-byte AS capable

    Send AIGP

    Send multicast attributes

     Multiple label stack: Enabled
```

```
/* Verify that the multiple label stack is enabled. */
Router# show bgp labels

...

Status codes: s suppressed, d damped, h history, * valid, > best

              i - internal, r RIB-failure, S stale, N Nexthop-discard

Origin codes: i - IGP, e - EGP, ? - incomplete

   Network            Next Hop        Rcvd Label      Local Label

*>i10.1.1.1/32         10.3.2.2         14001 16001     24193

                                      32001

*>i1.2.2.2/32         10.4.3.1        15002 17002     24199

                                       32002

*>i1.3.3.3/32         10.3.2.2         14001 16001     24200

                                      32002

...

/* */

Router# show route 10.1.1.1/32 detail

Routing entry for 10.1.1.1/32

  Known via "bgp 100", distance 200, metric 476387081, [ei]-bgp, labeled unicast (3107)

...

 Routing Descriptor Blocks

    209.165.201.1, from 10.0.1.101

      Route metric is 476387081

      Labels: 0x36b1 0x3e81 0x7d01 (14001 16001 32001)

      Tunnel ID: None

      Binding Label: None

      Extended communities count: 0

      NHID:0x0(Ref:0)

      MPLS eid:0x1380b00000003


/*  Verify that the multiple label stack is enabled. */

Router# show cef 10.1.1.1/32 detail
```

```
10.1.1.1/32, version 251579, internal 0x5000001 0x0 (ptr 0xa0241200) [1], 0x0 (0xa03feab8),
 0xa08

(0x9fced2b0)

 ...

   via 10.3.2.2/32, 3 dependencies, recursive [flags 0x6000]

    path-idx 0 NHID 0x0 [0x9e873ca0 0x0]

    recursion-via-/32

    next hop 10.3.2.2/32 via 24192/0/21

     local label 24193

     next hop 10.3.2.2/32 Te0/0/0/0/1  labels imposed {ImplNull 14001 16001 32001}


/* Verify the maximum supported depth of the label stack. If the number of labels received
 exceeds the maximum
supported by the platform, the prefix is not downloaded to the RIB and hence routing issues
 may occur. */

Router# show bgp ipv4 labeled-unicast process performance detail

...

Address Family: IPv4 Labeled-unicast

State: Normal mode.

BGP Table Version: 177675

Attribute download: Disabled

ASBR functionality enabled

Label retention timer value 5 mins

Soft Reconfig Entries: 367

Table bit-field size : 1 Chunk element size : 3

Maximum supported label-stack depth:

   For IPv4 Nexthop: 3

   For IPv6 Nexthop: 0

...
```

# iBGP Multipath Load Sharing

When a Border Gateway Protocol (BGP) speaking router that has no local policy configured, receives multiple network layer reachability information (NLRI) from the internal BGP (iBGP) for the same destination, the router will choose one iBGP path as the best path. The best path is then installed in the IP routing table of the router. The iBGP Multipath Load Sharing feature enables the BGP speaking router to select multiple iBGP paths as the best paths to a destination. The best paths or multipaths are then installed in the IP routing table of the router.

iBGP Multipath Load Sharing Reference, on page 142 provides additional details.

## Configure iBGP Multipath Load Sharing

Perform this task to configure the iBGP Multipath Load Sharing:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** {**ipv4**|**ipv6**} {**unicast**|**multicast**}

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 multicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

**Step 4** **maximum-paths ibgp** *number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# maximum-paths ibgp 30
```

Configures the maximum number of iBGP paths for load sharing.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### iBGP Multipath Loadsharing Configuration: Example

The following is a sample configuration where 30 paths are used for loadsharing:

```
router bgp 100
 address-family ipv4 multicast
  maximum-paths ibgp 30
 !
!
end
```

# Route Dampening

Route dampening is a BGP feature that minimizes the propagation of flapping routes across an internetwork. A route is considered to be flapping when it is repeatedly available, then unavailable, then available, then unavailable, and so on.

For example, consider a network with three BGP autonomous systems: autonomous system 1, autonomous system 2, and autonomous system 3. Suppose the route to network A in autonomous system 1 flaps (it becomes unavailable). Under circumstances without route dampening, the eBGP neighbor of autonomous system 1 to autonomous system 2 sends a withdraw message to autonomous system 2. The border router in autonomous system 2, in turn, propagates the withdrawal message to autonomous system 3. When the route to network A reappears, autonomous system 1 sends an advertisement message to autonomous system 2, which sends it to autonomous system 3. If the route to network A repeatedly becomes unavailable, then available, many withdrawal and advertisement messages are sent. Route flapping is a problem in an internetwork connected to the Internet, because a route flap in the Internet backbone usually involves many routes.

The route dampening feature minimizes the flapping problem as follows. Suppose again that the route to network A flaps. The router in autonomous system 2 (in which route dampening is enabled) assigns network A a penalty of 1000 and moves it to history state. The router in autonomous system 2 continues to advertise the status of the route to neighbors. The penalties are cumulative. When the route flaps so often that the penalty exceeds a configurable suppression limit, the router stops advertising the route to network A, regardless of how many times it flaps. Thus, the route is dampened.

The penalty placed on network A is decayed until the reuse limit is reached, upon which the route is once again advertised. At half of the reuse limit, the dampening information for the route to network A is removed.

**Note** No penalty is applied to a BGP peer reset when route dampening is enabled, even though the reset withdraws the route.

## Configuring BGP Route Dampening

Perform this task to configure and monitor BGP route dampening.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **bgp dampening** [ *half-life* [ *reuse suppress max-suppress-time* ] | **route-policy** *route-policy-name* ]

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# bgp dampening 30 1500 10000 120
```

Configures BGP dampening for the specified address family.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Routing Policy Enforcement

External BGP (eBGP) neighbors must have an inbound and outbound policy configured. If no policy is configured, no routes are accepted from the neighbor, nor are any routes advertised to it. This added security measure ensures that routes cannot accidentally be accepted or advertised in the case of a configuration omission error.

**Note** This enforcement affects only eBGP neighbors (neighbors in a different autonomous system than this router). For internal BGP (iBGP) neighbors (neighbors in the same autonomous system), all routes are accepted or advertised if there is no policy.

## Apply Policy When Updating Routing Table

The table policy feature in BGP allows you to configure traffic index values on routes as they are installed in the global routing table. This feature is enabled using the table-policy command and supports the BGP policy accounting feature. Table policy also provides the ability to drop routes from the RIB based on match criteria. This feature can be useful in certain applications and should be used with caution as it can easily create a routing traffic drop where BGP advertises routes to neighbors that BGP does not install in its global routing table and forwarding table.

Perform this task to apply a routing policy to routes being installed into the routing table.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120.6
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **table-policy** *policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# table-policy tbl-plcy-A
```

Applies the specified policy to routes being installed into the routing table.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Applying routing policy: Example

In the following example, for an eBGP neighbor, if all routes should be accepted and advertised with no modifications, a simple pass-all policy is configured:

```
RP/0/RP0/CPU0:router(config)# route-policy pass-all
RP/0/RP0/CPU0:router(config-rpl)# pass
RP/0/RP0/CPU0:router(config-rpl)# end-policy
RP/0/RP0/CPU0:router(config)# commit
```

Use the **route-policy (BGP)** command in the neighbor address-family configuration mode to apply the pass-all policy to a neighbor. The following example shows how to allow all IPv4 unicast routes to be received from neighbor 192.168.40.42 and advertise all IPv4 unicast routes back to it:

```
RP/0/RP0/CPU0:router(config)# router bgp 1
RP/0/RP0/CPU0:router(config-bgp)# neighbor 192.168.40.24
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 21
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy pass-all in
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy pass-all out
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# commit
```

Use the **show bgp summary** command to display eBGP neighbors that do not have both an inbound and outbound policy for every active address family. In the following example, such eBGP neighbors are indicated in the output with an exclamation (!) mark:

```
RP/0/RP0/CPU0:router# show bgp all all summary

Address Family: IPv4 Unicast
============================

BGP router identifier 10.0.0.1, local AS number 1
BGP generic scan interval 60 secs
BGP main routing table version 41
BGP scan interval 60 secs
BGP is operating in STANDALONE mode.

Process         RecvTblVer    bRIB/RIB  SendTblVer
Speaker                 41          41          41

Neighbor        Spk    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  St/PfxRcd
10.0.101.1        0     1     919     925       41    0    0 15:15:08        10
10.0.101.2        0     2       0       0        0    0    0 00:00:00 Idle
```

# Configure BGP Neighbor Group and Neighbors

Perform this task to configure BGP neighbor groups and apply the neighbor group configuration to a neighbor. A neighbor group is a template that holds address family-independent and address family-dependent configurations that are associated with the neighbor.

After a neighbor group is configured, each neighbor can inherit the configuration through the **use**command. If a neighbor is configured to use a neighbor group, the neighbor (by default) inherits the entire configuration of the neighbor group, which includes the address family-independent and address family-dependent configurations. The inherited configuration can be overridden if you directly configure commands for the neighbor or configure session groups or address family groups through the **use**command.

You can configure an address family-independent configuration under the neighbor group. An address family-dependent configuration requires you to configure the address family under the neighbor group to enter address family submode. From neighbor group configuration mode, you can configure address family-independent parameters for the neighbor group. Use the **address-family**command when in the neighbor group configuration mode. After specifying the neighbor group name using the **neighbor group** command, you can assign options to the neighbor group.

**Note** All commands that can be configured under a specified neighbor group can be configured under a neighbor.

**Note** In Cisco IOS-XR versions prior to 6.3.2, you cannot remove a autonomous system that belongs to a BGP neighbor and move it under a BGP neigbhorgroup using a single IOS-XR commit. Effective with 6.3.2, you can move the autonoums system from a neighbor to a neighbor group in a single IOS-XR commit.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** exit

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# exit
```

Exits the current configuration mode.

**Step 5** **neighbor-group** *name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor-group nbr-grp-A
```

Places the router in neighbor group configuration mode.

**Step 6** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbrgrp)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 7** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbrgrp)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 8** **route-policy** *route-policy-name* { **in** | **out** }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbrgrp-af)# route-policy drop-as-1234 in
```

(Optional) Applies the specified policy to inbound IPv4 unicast routes.

**Step 9** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbrgrp-af)# exit
```

Exits the current configuration mode.

**Step 10** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbrgrp)# exit
```

Exits the current configuration mode.

**Step 11** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 12** **use neighbor-group** *group-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# use neighbor-group nbr-grp-A
```

(Optional) Specifies that the BGP neighbor inherit configuration from the specified neighbor group.

**Step 13** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 14** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### BGP Neighbor Configuration: Example

The following example shows how BGP neighbors on an autonomous system are configured to share information. In the example, a BGP router is assigned to autonomous system 109, and two networks are listed as originating in the autonomous system. Then the addresses of three remote routers (and their autonomous systems) are listed. The router being configured shares information about networks 172 .16 .0.0 and 192.168 .7.0 with the neighbor routers. The first router listed is in a different autonomous system; the second **neighbor** and **remote-as** commands specify an internal neighbor (with the same autonomous system number) at address 172 .26 .234.2; and the third **neighbor** and **remote-as** commands specify a neighbor on a different autonomous system.

```
route-policy pass-all
 pass
end-policy
router bgp 109
 address-family ipv4 unicast
  network 172.16.0.0 255.255.0.0
  network 192.168.7.0 255.255.0.0
  neighbor 172.16.200.1
```

```
  remote-as 167

address-family ipv4 unicast
 route-policy pass-all in
 route-policy pass-out out
 neighbor 172.26.234.2
  remote-as 109

address-family ipv4 unicast
 neighbor 172.26.64.19
  remote-as 99

address-family ipv4 unicast
 route-policy pass-all in
 route-policy pass-all out
```

# Disable BGP Neighbor

Perform this task to administratively shut down a neighbor session without removing the configuration.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 127
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **shutdown**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# shutdown
```

Disables all active sessions for the specified neighbor.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Resetting Neighbors Using BGP Inbound Soft Reset

Perform this task to trigger an inbound soft reset of the specified address families for the specified group or neighbors. The group is specified by the ***** , *ip-address* , *as-number* , or **external** keywords and arguments.

Resetting neighbors is useful if you change the inbound policy for the neighbors or any other configuration that affects the sending or receiving of routing updates. If an inbound soft reset is triggered, BGP sends a REFRESH request to the neighbor if the neighbor has advertised the ROUTE_REFRESH capability. To determine whether the neighbor has advertised the ROUTE_REFRESH capability, use the **show bgp neighbors** command.

### Procedure

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **show bgp neighbors**<br>**Example:**<br>`RP/0/RP0/CPU0:router# show bgp neighbors` | Verifies that received route refresh capability from the neighbor is enabled. |
| **Step 2** | **soft** [ **in** [ **prefix-filter** ] \| **out** ]<br>**Example:**<br>`RP/0/RP0/CPU0:router# clear bgp ipv4 unicast 10.0.0.1 soft in` | Soft resets a BGP neighbor.<br>• The ***** keyword resets all BGP neighbors.<br>• The *ip-address* argument specifies the address of the neighbor to be reset.<br>• The *as-number* argument specifies that all neighbors that match the autonomous system number be reset.<br>• The **external** keyword specifies that all external neighbors are reset. |

## Resetting Neighbors Using BGP Outbound Soft Reset

Perform this task to trigger an outbound soft reset of the specified address families for the specified group or neighbors. The group is specified by the ***** , *ip-address* , *as-number* , or **external** keywords and arguments.

Resetting neighbors is useful if you change the outbound policy for the neighbors or any other configuration that affects the sending or receiving of routing updates.

If an outbound soft reset is triggered, BGP resends all routes for the address family to the given neighbors.

To determine whether the neighbor has advertised the ROUTE_REFRESH capability, use the **show bgp neighbors** command.

### Procedure

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **show bgp neighbors**<br>**Example:**<br>`RP/0/RP0/CPU0:router# show bgp neighbors` | Verifies that received route refresh capability from the neighbor is enabled. |
| **Step 2** | **Example:**<br>`RP/0/RP0/CPU0:router# clear bgp ipv4 unicast 10.0.0.2 soft out` | Soft resets a BGP neighbor.<br>• The ***** keyword resets all BGP neighbors.<br>• The *ip-address* argument specifies the address of the neighbor to be reset.<br>• The *as-number* argument specifies that all neighbors that match the autonomous system number be reset.<br>• The **external** keyword specifies that all external neighbors are reset. |

## Reset Neighbors Using BGP Hard Reset

Perform this task to reset neighbors using a hard reset. A hard reset removes the TCP connection to the neighbor, removes all routes received from the neighbor from the BGP table, and then re-establishes the session with the neighbor. If the **graceful** keyword is specified, the routes from the neighbor are not removed from the BGP table immediately, but are marked as stale. After the session is re-established, any stale route that has not been received again from the neighbor is removed.

### Procedure

**clear bgp** { **ipv4** { **unicast** | **labeled-unicast** | **all** | **tunnel** **tunnel** | **mdt** } | **ipv6** **unicast** | **all** | **labeled-unicast** } | **all** { **unicast** | **multicast** | **all** | **labeled-unicast** | **mdt** | **tunnel** } | **vpnv4 unicast** | **vrf** { *vrf-name* | **all** } { **ipv4** **unicast** | **labeled-unicast** } | **ipv6 unicast** } | **vpnv6 unicast** } { ***** | *ip-address* | **as** *as-number* | **external** } [ **graceful** ] **soft** [ **in** [ **prefix-filter** ] | **out** ] **clear bgp** { **ipv4** | **ipv6**} { **unicast** | **labeled-unicast** }

**Example:**

```
RP/0/RP0/CPU0:router# clear bgp ipv4 unicast 10.0.0.3 long-lived-stale
```

Clears a BGP neighbor.

- The ***** keyword resets all BGP neighbors.
- The *ip-address* argument specifies the address of the neighbor to be reset.

- The *as-number* argument specifies that all neighbors that match the autonomous system number be reset.
- The **external** keyword specifies that all external neighbors are reset.

The **graceful** keyword specifies a graceful restart.

## Configure Software to Store Updates from Neighbor

Perform this task to configure the software to store updates received from a neighbor.

The **soft-reconfiguration inbound** command causes a route refresh request to be sent to the neighbor if the neighbor is route refresh capable. If the neighbor is not route refresh capable, the neighbor must be reset to relearn received routes using the **clear bgp soft** command.

**Note** Storing updates from a neighbor works only if either the neighbor is route refresh capable or the **soft-reconfiguration inbound** command is configured. Even if the neighbor is route refresh capable and the **soft-reconfiguration inbound** command is configured, the original routes are not stored unless the **always** option is used with the command. The original routes can be easily retrieved with a route refresh request. Route refresh sends a request to the peer to resend its routing information. The **soft-reconfiguration inbound** command stores all paths received from the peer in an unmodified form and refers to these stored paths during the clear. Soft reconfiguration is memory intensive.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 5** **soft-reconfiguration inbound** [ **always]**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# soft-reconfiguration inbound always
```

Configures the software to store updates received from a specified neighbor. Soft reconfiguration inbound causes the software to store the original unmodified route in addition to a route that is modified or filtered. This allows a "soft clear" to be performed after the inbound policy is changed.

Soft reconfiguration enables the software to store the incoming updates before apply policy if route refresh is not supported by the peer (otherwise a copy of the update is not stored). The **always** keyword forces the software to store a copy even when route refresh is supported by the peer.

**Step 6** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Log Neighbor Changes

Logging neighbor changes is enabled by default. Use the**bgp log neighbor changes disable** command to turn off logging. Use the **no bgp log neighbor changes disable** command to turn logging back on, if it has been disabled.

```
RP/0/RP0/CPU0:(config)# router bgp 100
RP/0/RP0/CPU0:(config-bgp)# bgp log neighbor changes disable

RP/0/RP0/CPU0:(config)# router bgp 100
RP/0/RP0/CPU0:(config-bgp)# no bgp log neighbor changes disable
```

# BGP Route Reflectors

BGP requires that all iBGP speakers be fully meshed. However, this requirement does not scale well when there are many iBGP speakers. Instead of configuring a confederation, you can reduce the iBGP mesh by using a route reflector configuration. With route reflectors, all iBGP speakers need not be fully meshed because there is a method to pass learned routes to neighbors. In this model, an iBGP peer is configured to be a route reflector responsible for passing iBGP learned routes to a set of iBGP neighbors.

In #unique_51 unique_51_Connect_42_fig_3980C23832D84D43BB58222F040E5A96 , Router B is configured as a route reflector. When the route reflector receives routes advertised from Router A, it advertises them to Router C, and vice versa. This scheme eliminates the need for the iBGP session between routers A and C.

See BGP Route Reflectors Reference, on page 141 for additional details on route reflectors.

## Configure Route Reflector for BGP

Perform this task to configure a route reflector for BGP.

All the neighbors configured with the **route-reflector-client**command are members of the client group, and the remaining iBGP peers are members of the nonclient group for the local route reflector.

Together, a route reflector and its clients form a *cluster*. A cluster of clients usually has a single route reflector. In such instances, the cluster is identified by the software as the router ID of the route reflector. To increase redundancy and avoid a single point of failure in the network, a cluster can have more than one route reflector. If it does, all route reflectors in the cluster must be configured with the same 4-byte cluster ID so that a route reflector can recognize updates from route reflectors in the same cluster. The **bgp cluster-id** command is used to configure the cluster ID when the cluster has more than one route reflector.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **bgp cluster-id** *cluster-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp cluster-id 192.168.70.1
```

Configures the local router as one of the route reflectors serving the cluster. It is configured with a specified cluster ID to identify the cluster.

**Step 4** **neighbor** *ip-address*

**Example:**

```
  RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 5** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2003
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 6** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-nbr)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 7** **route-reflector-client**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-reflector-client
```

Configures the router as a BGP route reflector and configures the neighbor as its client.

**Step 8** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### BGP Route Reflector: Example

The following example shows how to use an address family to configure internal BGP peer 10.1.1.1 as a route reflector client for unicast prefixes:

```
router bgp 140
 address-family ipv4 unicast
  neighbor 10.1.1.1
   remote-as 140
   address-family ipv4 unicast
    route-reflector-client
    exit
```

# Configure BGP Route Filtering by Route Policy

Perform this task to configure BGP routing filtering by route policy.

**Procedure**

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **configure**<br>**Example:**<br>`RP/0/RP0/CPU0:router# configure` | Enters mode. |
| **Step 2** | **route-policy** *name*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config)# route-policy drop-as-1234`<br>`RP/0/RP0/CPU0:router(config-rpl)# if as-path passes-through '1234' then`<br>`RP/0/RP0/CPU0:router(config-rpl)# apply check-communities`<br>`RP/0/RP0/CPU0:router(config-rpl)# else`<br>`RP/0/RP0/CPU0:router(config-rpl)# pass`<br>`RP/0/RP0/CPU0:router(config-rpl)# endif` | (Optional) Creates a route policy and enters route policy configuration mode, where you can define the route policy. |
| **Step 3** | **end-policy**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-rpl)# end-policy` | (Optional) Ends the definition of a route policy and exits route policy configuration mode. |
| **Step 4** | **router bgp** *as-number*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config)# router bgp 120` | Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process. |
| **Step 5** | **neighbor** *ip-address*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24` | Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer. |
| **Step 6** | **address-family** { **ipv4** \| **ipv6** } **unicast**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast` | Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.<br>To see a list of all the possible keywords and arguments for this command, use the CLI help (?). |
| **Step 7** | **route-policy** *route-policy-name* { **in** \| **out** } | Applies the specified policy to inbound routes. |

| | Command or Action | Purpose |
|---|---|---|
| | **Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy drop-as-1234 in` | |
| **Step 8** | Use the **commit** or **end** command. | **commit** —Saves the configuration changes and remains within the configuration session.<br>**end** —Prompts user to take one of these actions:<br>• **Yes** — Saves configuration changes and exits the configuration session.<br>• **No** —Exits the configuration session without committing the configuration changes.<br>• **Cancel** —Remains in the configuration session, without committing the configuration changes. |

# Configure BGP Attribute Filtering

The BGP Attribute Filter checks integrity of BGP updates in BGP update messages and optimizes reaction when detecting invalid attributes. BGP Update message contains a list of mandatory and optional attributes. These attributes in the update message include MED, LOCAL_PREF, COMMUNITY, and so on. In some cases, if the attributes are malformed, there is a need to filter these attributes at the receiving end of the router. The BGP Attribute Filter functionality filters the attributes received in the incoming update message. The attribute filter can also be used to filter any attributes that may potentially cause undesirable behavior on the receiving router. Some of the BGP updates are malformed due to wrong formatting of attributes such as the network layer reachability information (NLRI) or other fields in the update message. These malformed updates, when received, causes undesirable behavior on the receiving routers. Such undesirable behavior may be encountered during update message parsing or during re-advertisement of received NLRIs. In such scenarios, its better to filter these corrupted attributes at the receiving end.

The Attribute-filtering is configured by specifying a single or a range of attribute codes and an associated action. When a received Update message contains one or more filtered attributes, the configured action is applied on the message. Optionally, the Update message is also stored to facilitate further debugging and a syslog message is generated on the console. When an attribute matches the filter, further processing of the attribute is stopped and the corresponding action is taken. Perform the following tasks to configure BGP attribute filtering:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **attribute-filter group** *attribute-filter group name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# attribute-filter group ag_discard_med
```

Specifies the attribute-filter group name and enters the attribute-filter group configuration mode, allowing you to configure a specific attribute filter group for a BGP neighbor.

**Step 4** **attribute** *attribute code* { **discard** | **treat-as-withdraw** }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-attrfg)# attribute 24 discard
```

Specifies a single or a range of attribute codes and an associated action. The allowed actions are:

- Treat-as-withdraw— Considers the update message for withdrawal. The associated IPv4-unicast or MP_REACH NLRIs, if present, are withdrawn from the neighbor's Adj-RIB-In.
- Discard Attribute— Discards this attribute. The matching attributes alone are discarded and the rest of the Update message is processed normally.

# BGP Next Hop Tracking

BGP receives notifications from the Routing Information Base (RIB) when next-hop information changes (event-driven notifications). BGP obtains next-hop information from the RIB to:

- Determine whether a next hop is reachable.
- Find the fully recursed IGP metric to the next hop (used in the best-path calculation).
- Validate the received next hops.
- Calculate the outgoing next hops.
- Verify the reachability and connectedness of neighbors.

BGP Next Hop Reference, on page 137 provides additional conceptual details on BGP next hop.

## Configure BGP Next-Hop Trigger Delay

Perform this task to configure BGP next-hop trigger delay. The Routing Information Base (RIB) classifies the dampening notifications based on the severity of the changes. Event notifications are classified as critical and noncritical. This task allows you to specify the minimum batching interval for the critical and noncritical events.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **nexthop trigger-delay** { **critical** *delay* | **non-critical** *delay* }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# nexthop trigger-delay critical 15000
```

Sets the critical next-hop trigger delay.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Disable Next-Hop Processing on BGP Updates

Perform this task to disable next-hop calculation for a neighbor and insert your own address in the next-hop field of BGP updates. Disabling the calculation of the best next hop to use when advertising a route causes all routes to be advertised with the network device as the next hop.

**Note** Next-hop processing can be disabled for address family group, neighbor group, or neighbor address family.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 206
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 5** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 6** **next-hop-self**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# next-hop-self
```

Sets the next-hop attribute for all routes advertised to the specified neighbor to the address of the local router. Disabling the calculation of the best next hop to use when advertising a route causes all routes to be advertised with the local network device as the next hop.

**Step 7** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# BGP Cost Community

The BGP cost community is a nontransitive extended community attribute that is passed to internal BGP (iBGP) and confederation peers but not to external BGP (eBGP) peers. The cost community feature allows you to customize the local route preference and influence the best-path selection process by assigning cost values to specific routes. The extended community format defines generic points of insertion (POI) that influence the best-path decision at different points in the best-path algorithm.

BGP Cost Community Reference, on page 137 provides additional conceptual details on BGP cost community.

## Configure BGP Cost Community

BGP receives multiple paths to the same destination and it uses the best-path algorithm to decide which is the best path to install in RIB. To enable users to determine an exit point after partial comparison, the cost community is defined to tie-break equal paths during the best-path selection process. Perform this task to configure the BGP cost community.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **route-policy** *name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# route-policy costA
```

Enters route policy configuration mode and specifies the name of the route policy to be configured.

**Step 3** **set extcommunity cost** { *cost-extcommunity-set-name* | *cost-inline-extcommunity-set* } [ **additive** ]

**Example:**

```
RP/0/RP0/CPU0:router(config)# set extcommunity cost cost_A
```

Specifies the BGP extended community attribute for cost.

**Step 4** **end-policy**

**Example:**

```
RP/0/RP0/CPU0:router(config)# end-policy
```

Ends the definition of a route policy and exits route policy configuration mode.

**Step 5** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Enters BGP configuration mode allowing you to configure the BGP routing process.

**Step 6** Do one of the following:

- **default-information originate**
- **aggregate-address** *address/mask-length* [ **as-set** ] [ **as-confed-set** ] [ **summary-only** ] [ **route-policy** *route-policy-name* ]
- **redistribute connected** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- *process-id* [ **match** { **external** | **internal** }] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute isis** *process-id* [ **level** { **1** | **1-inter-area** | **2** }] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute ospf** *process-id* [ **match** { **external** [ **1** | **2** ] | **internal** | **nssa-external** [ **1** | **2** ]}] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]

Applies the cost community to the attach point (route policy).

**Step 7** Do one of the following:

- **redistribute ospfv3** *process-id* [ **match** { **external** [ **1** | **2** ] | **internal** | **nssa-external** [ **1** | **2** ]}] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute rip** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute static** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **network** { *ip-address/prefix-length* | *ip-address mask* } [ **route-policy** *route-policy-name* ]
- **neighbor** *ip-address* **remote-as** *as-number*
- **route-policy** *route-policy-name* { **in** | **out** }

**Step 8** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.

- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

**Step 9** **show bgp** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router# show bgp 172.168.40.24
```

Displays the cost community in the following format:

Cost: *POI* : *cost-community-ID* : *cost-number*

## Configure BGP Community and Extended-Community Advertisements

Perform this task to specify that community/extended-community attributes should be sent to an eBGP neighbor. These attributes are not sent to an eBGP neighbor by default. By contrast, they are always sent to iBGP neighbors. This section provides examples on how to enable sending community attributes. The **send-community-ebgp** keyword can be replaced by the **send-extended-community-ebgp** keyword to enable sending extended-communities.

If the **send-community-ebgp** command is configured for a neighbor group or address family group, all neighbors using the group inherit the configuration. Configuring the command specifically for a neighbor overrides inherited values.

**Note** BGP community and extended-community filtering cannot be configured for iBGP neighbors. Communities and extended-communities are always sent to iBGP neighbors under VPNv4, MDT, IPv4, and IPv6 address families.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 5** **address-family**{**ipv4** {**labeled-unicast** | **unicast** | **mdt** | | **mvpn** | **rt-filter** | **tunnel**} | **ipv6** {**labeled-unicast** | **mvpn** | **unicast**}}

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv6 unicast
```

Enters neighbor address family configuration mode for the specified address family. Use either **ipv4** or **ipv6** address family keyword with one of the specified address family sub mode identifiers.

IPv6 address family mode supports these sub modes:

- **labeled-unicast**
- **mvpn**
- **unicast**

IPv4 address family mode supports these sub modes:

- **labeled-unicast**
- **mdt**
- **mvpn**
- **rt-filter**
- **tunnel**
- **unicast**

**Step 6** Use one of these commands:

- **send-community-ebgp**
- **send-extended-community-ebgp**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# send-community-ebgp
```

or

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# send-extended-community-ebgp
```

Specifies that the router send community attributes or extended community attributes (which are disabled by default for eBGP neighbors) to a specified eBGP neighbor.

**Step 7** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Configuring BGP Large Communities

BGP communities provide a way to group destinations and apply routing decisions such as acceptance, rejection, preference, or redistribution on a group of destinations using community attributes. BGP community attributes are variable length attributes consisting of a set of one or more 4-byte values which are split into two parts of 16 bits. The higher-order 16 bits represents the AS number and the lower order bits represents a locally defined value assigned by the operator of the AS.

Since the adoption of 4-byte ASNs (RFC6793), the BGP communities attribute can no longer accommodate the 4 byte ASNs as you need more than 4 bytes to encode the 4-byte ASN and an AS specific value that you want to tag with the route. Although BGP extended community permits a 4-byte AS to be encoded as the global administrator field, the local administrator field has only 2-byte of available space. So, 6-byte extended community attribute is also unsuitable. To overcome this limitation, you can configure a 12-byte BGP large community which is an optional attribute that provides the most significant 4-byte value to encode autonomous system number as the global administrator and the remaining two 4-byte assigned numbers to encode the local values.

Similar to BGP communities, routers can apply BGP large communities to BGP routes by using route policy languages (RPL) and other routers can then perform actions based on the community that is attached to the route. The policy language provides sets as a container for groups of values for matching purposes.

When large communities are specified in other commands, they are specified as three non negative decimal integers separated by colons. For example, 1:2:3. Each integer is stored in 32 bits. The possible range for each integer is 0 to 4294967295.

In route-policy statements, each integer in the BGP large community can be replaced by any of the following expressions :

- [x..y] — This expression specifies a range between x and y, inclusive.
- * —This expression stands for any number.
- peeras — This expression is replaced by the AS number of the neigbhor from which the community is received or to which the community is sent, as appropriate.
- not-peeras —This expression matches any number other than the peeras.
- private-as — This expression specifies any number in the private ASN range: [64512..65534] and [4200000000..4294967294].

These expressions can be also used in policy-match statements.

IOS regular expression (ios-regex) and DFA style regular expression (dfa-regex) can be used in any of the large-community policy match and delete statements. For example, the IOS regular expression ios-regex '^5:.*:7$' is equivalent to the expression 5:*:7.

The **send-community-ebgp** command is extended to include BGP large communities. This command is required for the BGP speaker to send large communities to ebgp neighbors.

### Restrictions and Guidelines

The following restrictions and guidelines apply for BGP large communities:

- All functionalities of the BGP community attribute is available for the BGP large-community attribute.
- The **send-community-ebgp** command is required for the BGP speaker to send large communities to ebgp neighbors.
- There are no well-known large-communities.
- The peeras expression cannot be used in a large-community-set.
- The peeras expression can only be used in large-community match or delete statements that appear in route policies that are applied at the neighbor-in or neighbor-out attach points.
- The not-peeras expression cannot be used in a large-community-set or in policy set statements.

### Configuration Example: Large Community Set

A large-community set defines a set of large communities. Named large-community sets are used in route-policy match and set statements.

This example shows how to create a named large-community set.

```
RP/0/RP0/CPU0:router(config)# large-community-set catbert
RP/0/RP0/CPU0:router(config-largecomm)#  1: 2: 3,
RP/0/RP0/CPU0:router(config-largecomm)#  peeras:2:3
RP/0/RP0/CPU0:router(config-largecomm)# end-set
```

### Configuration Example: Set Large Community

The following example shows how to set the BGP large community attribute in a route, using the **set large-community** {*large-community-set-name* | *inline-large-community-set* | *parameter* } [**additive** ] command. You can specify a named large-community-set or an inline set. The **additive** keyword retains the large communities already present in the route and adds the new set of large communities. However the **additive** keyword does not result in duplicate entries.

If a particular large community is attached to a route and you specify the same large community again with the additive keyword in the set statement, then the specified large community is not added again. The merging operation removes duplicate entries. This also applies to the peeras keyword.

The peeras expression in the example is replaced by the AS number of the neighbor from which the BGP large community is received or to which the community is sent, as appropriate.

```
RP/0/RP0/CPU0:router(config)# route-policy mordac
RP/0/RP0/CPU0:router(config-rpl)# set large-community (1:2:3, peeras:2:3)
RP/0/RP0/CPU0:router(config-rpl)# end-set
RP/0/RP0/CPU0:router(config)# large-community-set catbert
RP/0/RP0/CPU0:router(config-largecomm)#  1: 2: 3,
RP/0/RP0/CPU0:router(config-largecomm)#  peeras:2:3
RP/0/RP0/CPU0:router(config-largecomm)# end-set
RP/0/RP0/CPU0:router(config)# route-policy wally
RP/0/RP0/CPU0:router(config-rpl)# set large-community catbert additive
RP/0/RP0/CPU0:router(config-rpl)# end-set
```

In this example, if the route-policy mordac is applied to a neighbor, the ASN of which is 1, then the large community (1:2:3) is set only once.

**Note** You should configure the **send-community-ebgp** command to send large communities to ebgp neighbors.

### Configuration Example: Large Community Matches-any

The following example shows how to configure a route policy to match any element of a large -community set. This is a boolean condition and returns true if any of the large communities in the route match any of the large communities in the match condition.

```
RP/0/RP0/CPU0:router(config)# route-policy elbonia
RP/0/RP0/CPU0:router(config-rpl)# if large-community matches-any (1:2:3, 4:5:*) then
RP/0/RP0/CPU0:router(config-rpl)#   set local-preference 94
RP/0/RP0/CPU0:router(config-rpl)#  endif
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

### Configuration Example: Large Community Matches-every

The following example shows how to configure a route policy where every match specification in the statement must be matched by at least one large community in the route.

```
RP/0/RP0/CPU0:router(config)# route-policy bob
RP/0/RP0/CPU0:router(config-rpl)# if large-community matches-every (*:*:3, 4:5:*) then
RP/0/RP0/CPU0:router(config-rpl)#   set local-preference 94
RP/0/RP0/CPU0:router(config-rpl)#  endif
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

In this example, routes with these sets of large communities return TRUE:

- (1:1:3, 4:5:10)
- (4:5:3) —This single large community matches both specifications.
- (1:1:3, 4:5:10, 7:6:5)

Routes with the following set of large communities return FALSE:

(1:1:3, 5:5:10)—The specification (4:5:*) is not matched.

### Configuration Example: Large Community Matches-within

The following example shows how to configure a route policy to match within a large community set. This is similar to the **large-community matches-any** command but every large community in the route must match at least one match specification. Note that if the route has no large communities, then it matches.

```
RP/0/RP0/CPU0:router(config)# route-policy bob
RP/0/RP0/CPU0:router(config-rpl)# if large-community matches-within (*:*:3, 4:5:*) then
RP/0/RP0/CPU0:router(config-rpl)#   set local-preference 103
RP/0/RP0/CPU0:router(config-rpl)#  endif
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

For example, routes with these sets of large communities return TRUE:

- (1:1:3, 4:5:10)
- (4:5:3)

- (1:2:3, 6:6:3, 9:4:3)

Routes with this set of large communities return FALSE:

(1:1:3, 4:5:10, 7:6:5) —The large community (7:6:5) does not match

### Configuration Example: Community Matches-within

The following example shows how to configure a route policy to match within the elements of a community set. This command is similar to the **community matches-any** command, but every community in the route must match at least one match specification. If the route has no communities, then it matches.

```
RP/0/RP0/CPU0:router(config)# route-policy bob
RP/0/RP0/CPU0:router(config-rpl)# if community matches-within (*:3, 5:*)  then
RP/0/RP0/CPU0:router(config-rpl)#   set local-preference 94
RP/0/RP0/CPU0:router(config-rpl)#  endif
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

For example, routes with these sets of communities return TRUE:

- (1:3, 5:10)
- (5:3)
- (2:3, 6:3, 4:3)

Routes with this set of communities return FALSE:

(1:3, 5:10, 6:5) —The community (6:5) does not match.

### Configuration Example: Large Community Is-empty

The following example shows using the **large-community is-empty** clause to filter routes that do not have the large-community attribute set.

```
RP/0/RP0/CPU0:router(config)# route-policy lrg_comm_rp4
RP/0/RP0/CPU0:router(config-rpl)# if large-community is-empty then
RP/0/RP0/CPU0:router(config-rpl)#   set local-preference 104
RP/0/RP0/CPU0:router(config-rpl)#  endif
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

### Configuration Example: Attribute Filter Group

The following example shows how to configure and apply the attribute-filter group with large-community attributes for a BGP neighbor. The filter specifies the BGP path attributes and an action to take when BGP update message is received. If an update message is received from the BGP neighbor that contains any of the specified attributes, then the specified action is taken. In this example, the attribute filter named dogbert is created and applied to the BGP neighbor 10.0.1.101. It specifies the large community attribute and the action of discard. That means, if the large community BGP path attribute is received in a BGP UPDATE message from the neighbor 10.0.1.101 then the attribute will be discarded before further processing of the message.

```
RP/0/RP0/CPU0:router(config)# router bgp 100
RP/0/RP0/CPU0:router(config-bgp)# attribute-filter group dogbert
RP/0/RP0/CPU0:router(config-bgp-attrfg)# attribute LARGE-COMMUNITY discard
RP/0/RP0/CPU0:router(config-bgp-attrfg)# neighbor 10.0.1.101
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 6461
RP/0/RP0/CPU0:router(config-bgp-nbr)# update in filtering
```

```
RP/0/RP0/CPU0:router(config-nbr-upd-filter)# attribute-filter group dogbert
```

### Configuration Example: Deleting Large Community

The following example shows how to delete specified BGP large-communities from a route policy using the **delete large-community** command.

```
RP/0/RP0/CPU0:router(config)# route-policy lrg_comm_rp2
RP/0/RP0/CPU0:router(config-rpl)# delete large-community in (ios-regex '^100000:')
RP/0/RP0/CPU0:router(config-rpl)# delete large-community all
RP/0/RP0/CPU0:router(config-rpl)# delete large-community not in (peeras:*:*, 41289:*:*)
```

### Verification

This example displays the routes with large-communities given in the **show bgp large-community** *list-of-large-communities* [**exact-match** ] command. If the optional keyword exact-match is used, then the listed routes will contain only the specified large communities. Otherwise, the displayed routes may contain additional large communities.

```
RP/0/0/CPU0:R1# show bgp large-community 1:2:3 5:6:7
Thu Mar 23 14:40:33.597 PDT
BGP router identifier 4.4.4.4, local AS number 3
BGP generic scan interval 60 secs
Non-stop routing is enabled
BGP table state: Active
Table ID: 0xe0000000   RD version: 66
BGP main routing table version 66
BGP NSR Initial initsync version 3 (Reached)
BGP NSR/ISSU Sync-Group versions 66/0
BGP scan interval 60 secs

Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop            Metric LocPrf Weight Path
* 10.0.0.3/32         10.10.10.3               0     94      0 ?
* 10.0.0.5/32         10.11.11.5               0             0 5 ?
```

This example displays the large community attached to a network using the **show bgp** *ip-address/ prefix-length* command.

```
RP/0/0/CPU0:R4# show bgp 10.3.3.3/32
Thu Mar 23 14:36:15.301 PDT
BGP routing table entry for 10.3.3.3/32
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 42          42
Last Modified: Mar 22 20:04:46.000 for 18:31:30
Paths: (1 available, best #1)
  Advertised to peers (in unique update groups):
    10.11.11.5
  Path #1: Received by speaker 0
  Advertised to peers (in unique update groups):
    10.11.11.5
  Local
    10.10.10.3 from 10.10.10.3 (10.3.3.3)
      Origin incomplete, metric 0, localpref 94, valid, internal, best, group-best
      Received Path ID 0, Local Path ID 0, version 42
      Community: 258:259 260:261 262:263 264:265
      Large Community: 1:2:3 5:6:7 4123456789:4123456780:4123456788
```

# Redistribute iBGP Routes into IGP

Perform this task to redistribute iBGP routes into an Interior Gateway Protocol (IGP), such as Intermediate System-to-Intermediate System (IS-IS) or Open Shortest Path First (OSPF).

**Note** Use of the **bgp redistribute-internal** command requires the **clear route *** command to be issued to reinstall all BGP routes into the IP routing table.

**Caution** Redistributing iBGP routes into IGPs may cause routing loops to form within an autonomous system. Use this command with caution.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **bgp redistribute-internal**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp redistribute-internal
```

Allows the redistribution of iBGP routes into an IGP, such as IS-IS or OSPF.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Redistribute IGPs to BGP

Perform this task to configure redistribution of a protocol into the VRF address family.

Even if Interior Gateway Protocols (IGPs) are used as the PE-CE protocol, the import logic happens through BGP. Therefore, all IGP routes have to be imported into the BGP VRF table.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **vrf** *vrf-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# vrf vrf_a
```

Enables BGP routing for a particular VRF on the PE router.

**Step 4** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 5** Do one of the following:

- **redistribute connected** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute isis** *process-id* [ **level** { **1** | **1-inter-area** | **2** }] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute ospf** *process-id* [ **match** { **external** [ **1** | **2** ] | **internal** | **nssa-external** [ **1** | **2** ]}] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute ospfv3** *process-id* [ **match** { **external** [ **1** | **2** ] | **internal** | **nssa-external** [ **1** | **2** ]}] [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute rip** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]
- **redistribute static** [ **metric** *metric-value* ] [ **route-policy** *route-policy-name* ]

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# redistribute ospf 1
```

Configures redistribution of a protocol into the VRF address family context.

The **redistribute** command is used if BGP is not used between the PE-CE routers. If BGP is used between PE-CE routers, the IGP that is used has to be redistributed into BGP to establish VPN connectivity with other PE sites. Redistribution is also required for inter-table import and export.

**Step 6** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Update Groups

The BGP Update Groups feature contains an algorithm that dynamically calculates and optimizes update groups of neighbors that share outbound policies and can share the update messages. The BGP Update Groups feature separates update group replication from peer group configuration, improving convergence time and flexibility of neighbor configuration.

## Monitor BGP Update Groups

This task displays information related to the processing of BGP update groups.

**Procedure**

**show bgp** [ **ipv4** { **unicast** | **multicast** | **all** | **tunnel** } | **ipv6** { **unicast** | **all** } | **all** { **unicast** | **multicast** | **all** **labeled-unicast** | **tunnel** } | **vpnv4 unicast** | **vrf** { *vrf-name* | **all** } [ **ipv4** **unicast** **ipv6 unicast** ] | **vpvn6 unicast** ] **update-group** [ **neighbor** *ip-address* | *process-id.index* [ **summary** | **performance-statistics** ]]

**Example:**

```
RP/0/RP0/CPU0:router# show bgp update-group 0.0
```

Displays information about BGP update groups.

- The *ip-address* argument displays the update groups to which that neighbor belongs.
- The *process-id.index* argument selects a particular update group to display and is specified as follows: process ID (dot) index. Process ID range is from 0 to 254. Index range is from 0 to 4294967295.
- The **summary** keyword displays summary information for neighbors in a particular update group.

- If no argument is specified, this command displays information for all update groups (for the specified address family).
- The **performance-statistics** keyword displays performance statistics for an update group.

### Displaying BGP Update Groups: Example

The following is sample output from the **show bgp update-group** command run in EXEC configurationXR EXEC mode:

```
show bgp update-group

 Update group for IPv4 Unicast, index 0.1:
   Attributes:
     Outbound Route map:rm
     Minimum advertisement interval:30
   Messages formatted:2, replicated:2
   Neighbors in this update group:
     10.0.101.92

 Update group for IPv4 Unicast, index 0.2:
   Attributes:
     Minimum advertisement interval:30
   Messages formatted:2, replicated:2
   Neighbors in this update group:
     10.0.101.91
```

# L3VPN iBGP PE-CE

The L3VPN iBGP PE-CE feature helps establish an iBGP (internal Border Gateway Protocol) session between the provider edge (PE) and customer edge (CE) devices to exchange BGP routing information. A BGP session between two BGP peers is said to be an iBGP session if the BGP peers are in the same autonomous systems.

## Restrictions for L3VPN iBGP PE-CE

The following restrictions apply to configuring L3VPN iBGP PE-CE:

- When the iBGP PE CE feature is toggled and the neighbor no longer supports route-refresh or soft-reconfiguration inbound, a manual session flap must be done to see the change. When this occurs, the following message is displayed:

  ```
  RP/0/RP0/CPU0: %ROUTING-BGP-5-CFG_CHG_RESET: Internal VPN client configuration change
  on neighbor 10.10.10.1 requires HARD reset
  (clear bgp 10.10.10.1) to take effect.
  ```

- iBGP PE CE CLI configuration is not available for peers under default-VRF, except for neighbor/session-group.
- This feature does not work on regular VPN clients (eBGP VPN clients).

- Attributes packed inside the ATTR_SET reflects changes made by the inbound route-policy on the iBGP CE and does not reflect the changes made by the export route-policy for the specified VRF.
- Different VRFs of the same VPN (that is, in different PE routers) that are configured with iBGP PE-CE peering sessions must use different Route Distinguisher (RD) values under respective VRFs. The iBGP PE CE feature does ot work if the RD values are the same for the ingress and egress VRF.

## Configuring L3VPN iBGP PE-CE

L3VPN iBGP PE-CE can be enabled on the neighbor, neighbor-group, or session-group. To configure L3VPN iBGP PE-CE, follow these steps:

### Before you begin

The CE must be an internal BGP peer.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **vrf** *vrf-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# vrf blue
```

Configures a VRF instance.

**Step 4** **neighbor** *ip-address* **internal-vpn-client**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf)# neighbor 10.0.0.0  internal-vpn-client
```

Configures a CE neighboring device with which to exchange routing information. The **neighbor internal-vpn-client** command stacks the iBGP-CE neighbor path in the VPN attribute set.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

**Step 6** **show bgp vrf** *vrf-name* **neighbors** *ip-address*

Displays whether the iBGP PE-CE feature is enabled for the VRF CE peer, or not.

**Step 7** **show bgp** { **vpnv4** | **vpnv6** } **unicast rd**

Displays the ATTR_SET attributes in the command output when the L3VPN iBGP PE-CE is enabled on a CE.

### Example

**Example: Configuring L3VPN iBGP PE-CE**

The following example shows how to configure L3VPN iBGP PE-CE:

```
R1(config-bgp-vrf-nbr)#neighbor 10.10.10.1 ?
. . .
 internal-vpn-client     Preserve iBGP CE neighbor path in ATTR_SET across VPN core
. . .
R1(config-bgp-vrf-nbr)#neighbor 10.10.10.1 internal-vpn-client
router bgp 65001
 bgp router-id 100.100.100.2
 address-family ipv4 unicast
address-family vpnv4 unicast
 !
 vrf ce-ibgp
  rd 65001:100
  address-family ipv4 unicast
  !
  neighbor 10.10.10.1
   remote-as 65001
   internal-vpn-client
```

The following is an example of the output of the **show bgp vrf** *vrf-name* **neighbors** *ip-address* command when the L3VPN iBGP PE-CE is enabled on a CE peer:

```
R1#show bgp vrf ce-ibgp neighbors 10.10.10.1
BGP neighbor is 10.10.10.1, vrf ce-ibgp
 Remote AS 65001, local AS 65001, internal link
 Remote router ID 100.100.100.1
  BGP state = Established, up for 00:00:19
  . . .
 Multi-protocol capability received
  Neighbor capabilities:
    Route refresh: advertised (old + new) and received (old + new)
    4-byte AS: advertised and received
    Address family IPv4 Unicast: advertised and received
CE attributes will be preserved across the core
  Received 2 messages, 0 notifications, 0 in queue
  Sent 2 messages, 0 notifications, 0 in queue
  . . .
```

The following is an example of the output of the **show bgp vpn4/vpn6 unicast rd** command when the L3VPN iBGP PE-CE is enabled on a CE peer:

```
BGP routing table entry for 1.1.1.0/24, Route Distinguisher: 200:300
Versions:
  Process           bRIB/RIB  SendTblVer
  Speaker                 10          10
Last Modified: Aug 28 13:11:17.000 for 00:01:00
Paths: (1 available, best #1)
  Advertised to update-groups (with more than one peer):
    0.2
Path #1: Received by speaker 0
  Advertised to update-groups (with more than one peer):
    0.2
  Local, (Received from a RR-client)
    20.20.20.2 from 20.20.20.2 (100.100.100.2)
      Received Label 24000
      Origin IGP, localpref 100, valid, internal, best, group-best, import-candidate,
      not-in-vrf Received Path ID 0, Local Path ID 1, version 10
      Extended community: RT:228:237
     ATTR-SET [
          Origin-AS: 200
          AS-Path: 51320 52325 59744 12947 21969 50346 18204 36304 41213
23906 33646
          Origin: incomplete
          Metric: 204
          Local-Pref: 234
          Aggregator: 304 34.3.3.3
          Atomic Aggregator
          Community: 1:60042 2:41661 3:47008 4:9280 5:39778 6:1069 7:15918
 8:8994 9:52701
10:10268 11:26276 12:8506 13:7131 14:65464 15:14304 16:33615 17:54991
18:40149 19:19401
          Extended community: RT:100:1 RT:1.1.1.1:1]
```

# Flow-tag propagation

The flow-tag propagation feature enables you to establish a co-relation between route-policies and user-policies. Flow-tag propagation using BGP allows user-side traffic-steering based on routing attributes such as, AS number, prefix lists, community strings and extended communities. Flow-tag is a logical numeric identifier that is distributed through RIB as one of the routing attribute of FIB entry in the FIB lookup table. A flow-tag is instantiated using the 'set' operation from RPL and is referenced in the C3PL PBR policy, where it is associated with actions (policy-rules) against the flow-tag value.

You can use flow-tag propagation to:

- Classify traffic based on destination IP addresses (using the Community number) or based on prefixes (using Community number or AS number).
- Select a TE-group that matches the cost of the path to reach a service-edge based on customer site service level agreements (SLA).
- Apply traffic policy (TE-group selection) for specific customers based on SLA with its clients.
- Divert traffic to application or cache server.

## Restrictions for Flow-Tag Propagation

Some restrictions are placed with regard to using Quality-of-service Policy Propagation Using Border Gateway Protocol (QPPB) and flow-tag feature together. These include:

- 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 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.

## Source and destination-based flow tag

The source-based flow tag feature allows you to match packets based on the flow-tag assigned to the source address of the incoming packets. Once matched, you can then apply any supported PBR action on this policy.

### Configure Source and Destination-based Flow Tag

This task applies flow-tag to a specified interface. The packets are matched based on the flow-tag assigned to the source address of the incoming packets.

**Note** You will not be able to enable both QPPB and flow tag feature simultaneously on an interface.

#### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **interface** *type interface-path-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-if)# interface
```

Enters interface configuration mode and associates one or more interfaces to the VRF.

**Step 3** **ipv4 | ipv6 bgp policy propagation input flow-tag**{**destination** | **source**}

**Example:**

```
RP/0/RP0/CPU0:router(config-if)# ipv4 bgp policy propagation input flow-tag source
```

Enables flow-tag policy propagation on source or destination IP address on an interface.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes, and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration mode, without committing the configuration changes.

### Example

The following show commands display outputs with PBR policy applied on the router:

```
show running-config interface gigabitEthernet 0/0/0/12
Thu Feb 12 01:51:37.820 UTC
interface GigabitEthernet0/0/0/12
 service-policy type pbr input flowMatchPolicy
 ipv4 bgp policy propagation input flow-tag source
 ipv4 address 192.5.1.2 255.255.255.0
!

RP/0/RSP0/CPU0:ASR9K-0#show running-config policy-map type pbr flowMatchPolicy
Thu Feb 12 01:51:45.776 UTC
policy-map type pbr flowMatchPolicy
 class type traffic flowMatch36
  transmit
 !
 class type traffic flowMatch38
  transmit
 !
 class type traffic class-default
 !
 end-policy-map
!

RP/0/RSP0/CPU0:ASR9K-0#show running-config class-map type traffic flowMatch36
Thu Feb 12 01:52:04.838 UTC
class-map type traffic match-any flowMatch36
 match flow-tag 36
 end-class-map
!
```

# BGP Keychains

BGP keychains enable keychain authentication between two BGP peers. The BGP endpoints must both comply with draft-bonica-tcp-auth-05.txt and a keychain on one endpoint and a password on the other endpoint does not work.

BGP is able to use the keychain to implement hitless key rollover for authentication. Key rollover specification is time based, and in the event of clock skew between the peers, the rollover process is impacted. The configurable tolerance specification allows for the accept window to be extended (before and after) by that margin. This accept window facilitates a hitless key rollover for applications (for example, routing and management protocols).

The key rollover does not impact the BGP session, unless there is a keychain configuration mismatch at the endpoints resulting in no common keys for the session traffic (send or accept).

# Configure Keychains for BGP

Keychains provide secure authentication by supporting different MAC authentication algorithms and provide graceful key rollover. Perform this task to configure keychains for BGP. This task is optional.

> **Note** If a keychain is configured for a neighbor group or a session group, a neighbor using the group inherits the keychain. Values of commands configured specifically for a neighbor override inherited values.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.168.40.24
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 2002
```

Creates a neighbor and assigns a remote autonomous system number to it.

**Step 5** **keychain** *name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# keychain kych_a
```

Configures keychain-based authentication.

**Step 6** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Master Key Tuple Configuration

This feature specifies TCP Authentication Option (TCP-AO), which replaces the TCP MD5 option. TCP-AO uses the Message Authentication Codes (MACs), which provides the following:

- Protection against replays for long-lived TCP connections
- More details on the security association with TCP connections than TCP MD5
- A larger set of MACs with minimal other system and operational changes

TCP-AO is compatible with Master Key Tuple (MKT) configuration. TCP-AO also protects connections when using the same MKT across repeated instances of a connection. TCP-AO protects the connections by using traffic key that are derived from the MKT, and then coordinates changes between the endpoints.

**Note** TCPAO and TCP MD5 are never permitted to be used simultaneously. TCP-AO supports IPv6, and is fully compatible with the proposed requirements for the replacement of TCP MD5.

Cisco provides the MKT configuration via the following configurations:

- keychain configuration
- tcp ao keychain configuration

The system translates each key, such “key_id” that is under a keychain, as MKT. The keychain configuration owns part of the configuration like secret, lifetimes, and algorithms. While the “tcp ao keychain” mode owns the TCP AO-specific configuration for an MKT (send_id and receive_id).

# Keychain Configurations

## Configuration Guidelines

In order to run a successful configuration, ensure that you follow the configuration guidelines:

- An allowed value range for both Send_ID and Receive_ID is 0 to 255.
- You can link only one keychain to an application neighbor.
- Under the same keychain, if you configure the same send_id key again under the keys that have an overlapping lifetime, then the old key becomes unusable until you correct the configuration.
- The system sends a warning message in the following scenarios:

    - If there is a change in Send_ID or Receive_ID.
    - If the corresponding key is currently active, and is in use by some connection.

- BGP neighbor can ONLY use one of the authentication options:
    - MD5
    - EA
    - AO

**Note** If you configure one of these options, the system rejects the other authentication options during the configuration time.

## Configuration Guidelines for TCP AO BGP Neighbor

The configuration guidelines are:

- Configure all the necessary configurations (key_string, MAC_algorithm, send_lifetime, accept_lifetime, send_id, receive_id) under key_id with the desired lifetime it wants to use the key_id for.
- Configure a matching MKT in the peer side with exactly same lifetime.
- Once a keychain-key is linked to tcp-ao, do not change the components of the key. If you want TCP to consider another key for use, you can configure that dynamically. Based on the 'start-time'of send lifetime, TCP AO uses the key.
- Send_ID and Receive_ID under a key_id (under a keychain) must have the same lifetime range. For example, send-lifetime==accept-lifetime.

  TCP considers only expiry of send-lifetime to transition to next active key and it does not consider accept-lifetime at all.
- Do not configure a key with send-lifetime that is covered by another key's send-lifetime.

  For example, if there is a key that is already configured with send-lifetime of "04:00:00 November 01, 2017 07:00:00 November 01, 2017" and the user now configures another key with send-lifetime of "05:00:00 November 01, 2017 06:00:00 November 01, 2017", this might result into connection flap.

  TCP AO tries to transition back to the old key once the new key is expired. However, if the new key has already expired, TCP AO can't use it, which might result in segment loss and hence connection flap.
- Configure minimum of 15 minutes of overlapping time between the two overlapping keys. When a key expires, TCP does not use it and hence out-of-order segments with that key are dropped.
- We recommend configuring send_id and receive_id to be same for a key_id for simplicity.
- TCP does not have any restriction on the number of keychains and keys under a keychain. The system does not support more than 4000 keychains, any number higher than 4000 might result in unexpected behaviors.

## Keychain Configuration

```
key chain <keychain_name>
   key <key_id>
      accept-lifetime <start-time> <end-time>
      key-string <master-key>
      send-lifetime <start-time> <end-time>
      cryptographic-algorithm <algorithm>
   !
!
```

## TCP Configuration

TCP provides a new tcp ao submode that specifies SendID and ReceiveID per key_id per keychain.

```
tcp ao
    keychain <keychain_name1>
        key-id <key_id> send_id <0-255> receive_id <0-255>
        !
```

Example:

```
tcp ao
 keychain bgp_ao
  key 0 SendID 0 ReceiveID 0
  key 1 SendID 1 ReceiveID 1
  key 2 SendID 3 ReceiveID 4
 !
 keychain ldp_ao
  key 1 SendID 100 ReceiveID 200
  key 120 SendID 1 ReceiveID 1
 !
```

## BGP Configurations

Applications like BGP provide the tcp-ao keychain and related information that it uses per neighbor. Following are the optional configurations per tcp-ao keychain:

- include-tcp-options
- accept-non-ao-connections

```
router bgp <AS-number>
neighbor <neighbor-ip>
  remote-as <remote-as-number>
  ao <keychain-name> include-tcp-options enable/disable  <accept-ao-mismatch-connections>
!
```

## XML Configurations

### BGP XML

### TCP-AO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
 <Set>
  <Configuration>
   <IP_TCP>
    <AO>
      <Enable>
```

```
          true
        </Enable>
        <KeychainTable>
          <Keychain>
           <Naming>
            <Name> bgp_ao_xml </Name>
           </Naming>
           <Enable>
            true
           </Enable>
            <KeyTable>
             <Key>
              <Naming>
               <KeyID> 0 </KeyID>
              </Naming>
               <SendID> 0 </SendID>
               <ReceiveID> 0 </ReceiveID>
             </Key>
            </KeyTable>
          </Keychain>
        </KeychainTable>
      </AO>
     </IP_TCP>
    </Configuration>
   </Set>
   <Commit/>
  </Request>
```

# BGP Nonstop Routing

The Border Gateway Protocol (BGP) Nonstop Routing (NSR) with Stateful Switchover (SSO) feature enables all bgp peerings to maintain the BGP state and ensure continuous packet forwarding during events that could interrupt service. Under NSR, events that might potentially interrupt service are not visible to peer routers. Protocol sessions are not interrupted and routing states are maintained across process restarts and switchovers.

BGP Nonstop Routing Reference, on page 139 for additional details.

## Configure BGP Nonstop Routing

BGP Nonstop Routing (BGP NSR) is enabled by default. If BGP NSR is disabled, use the **no nsr disable** command to turn BGP NSR back on.

**Note** In some scenarios, it is possible that some or all bgp sessions are not NSR-READY. The `show redundancy` command may still show that the bgp sessions are NSR-ready. Hence, we recommend that you verify the bgp nsr state by using the `show bgp sessions` command.

## Disable BGP Nonstop Routing

Perform this task to disable BGP Nonstop Routing (NSR):

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the BGP AS number, and enters the BGP configuration mode, for configuring BGP routing processes.

**Step 3** **nsr disable**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# nsr disable
```

Disables BGP Nonstop routing.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Disable BGP Nonstop Routing: Example

The following example shows how to disable BGP NSR:

```
configure
router bgp 120
no nsr
end
```

## Re-enable BGP Nonstop Routing

If BGP Nonstop Routing (NSR) is disabled, use the following steps to turn BGP NSR back on using the following steps:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the BGP AS number, and enters the BGP configuration mode, for configuring BGP routing processes.

**Step 3** **no nsr disable**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# nsr disable
```

Enables BGP Nonstop routing.

**Step 4** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Re-enable BGP Nonstop Routing: Example

The following example shows how to enable BGP NSR:

```
configure
router bgp 120
nsr
end
```

# Accumulated Interior Gateway Protocol Attribute

The Accumulated Interior Gateway Protocol (AiGP)Attribute is an optional non-transitive BGP Path Attribute. The attribute type code for the AiGP Attribute is to be assigned by IANA. The value field of the AiGP Attribute is defined as a set of Type/Length/Value elements (TLVs). The AiGP TLV contains the Accumulated IGP Metric.

The AiGP feature is required in the 3107 network to simulate the current OSPF behavior of computing the distance associated with a path. OSPF/LDP carries the prefix/label information only in the local area. Then, BGP carries the prefix/lable to all the remote areas by redistributing the routes into BGP at area boundaries. The routes/labels are then advertised using LSPs. The next hop for the route is changed at each ABR to local router which removes the need to leak OSPF routes across area boundaries. The bandwidth available on each of the core links is mapped to OSPF cost, hence it is imperative that BGP carries this cost correctly between each of the PEs. This functionality is achieved by using the AiGP.

## Originate Prefixes with AiGP

Perform this task to configure origination of routes with the AiGP metric:

### Before you begin

Origination of routes with the accumulated interior gateway protocol (AiGP) metric is controlled by configuration. AiGP attributes are attached to redistributed routes that satisfy following conditions:

- The protocol redistributing the route is enabled for AiGP.
- The route is an interior gateway protocol (iGP) route redistributed into border gateway protocol (BGP). The value assigned to the AiGP attribute is the value of iGP next hop to the route or as set by a route-policy.
- The route is a static route redistributed into BGP. The value assigned is the value of next hop to the route or as set by a route-policy.
- The route is imported into BGP through network statement. The value assigned is the value of next hop to the route or as set by a route-policy.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **route-policy** *aigp_policy*

**Example:**

```
RP/0/RP0/CPU0:router(config)# route-policy aip_policy
```

Enters route-policy configuration mode and sets the route-policy

**Step 3** **set aigp-metricigp-cost**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# set aigp-metric igp-cost
```

Sets the internal routing protocol cost as the aigp metric.

**Step 4** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# exit
```

Exits route-policy configuration mode.

**Step 5** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 6** **address-family** {**ipv4** | **ipv6**} **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

**Step 7** **redistribute ospf** *osp* **route-policy** *plcy_name***metric** *value*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)#redistribute ospf osp route-policy aigp_policy metric
1
```

Allows the redistribution of AiBGP metric into OSPF.

**Step 8** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Originating Prefixes With AiGP: Example

The following is a sample configuration for originating prefixes with the AiGP metric attribute:

```
route-policy aigp-policy
  set aigp-metric 4
  set aigp-metric igp-cost
end-policy
!
router bgp 100
 address-family ipv4 unicast
  network 10.2.3.4/24 route-policy aigp-policy
  redistribute ospf osp1 metric 4 route-policy aigp-policy
 !
!
end
```

# Configure BGP Accept Own

The BGP Accept Own feature allows you to handle self-originated VPN routes, which a BGP speaker receives from a route-reflector (RR). A 'self-originated' route is one which was originally advertized by the speaker itself. As per BGP protocol [RFC4271], a BGP speaker rejects advertisements that were originated by the speaker itself. However, the BGP Accept Own mechanism enables a router to accept the prefixes it has advertised, when reflected from a route-reflector that modifies certain attributes of the prefix. A special community called ACCEPT-OWN is attached to the prefix by the route-reflector, which is a signal to the receiving router to bypass the ORIGINATOR_ID and NEXTHOP/MP_REACH_NLRI check. Generally, the BGP speaker detects prefixes that are self-originated through the self-origination check (ORIGINATOR_ID, NEXTHOP/MP_REACH_NLRI) and drops the received updates. However, with the Accept Own community present in the update, the BGP speaker handles the route.

One of the applications of BGP Accept Own is auto-configuration of extranets within MPLS VPN networks. In an extranet configuration, routes present in one VRF is imported into another VRF on the same PE. Normally, the extranet mechanism requires that either the import-rt or the import policy of the extranet VRFs be modified to control import of the prefixes from another VRF. However, with Accept Own feature, the route-reflector can assert that control without the need for any configuration change on the PE. This way, the Accept Own feature provides a centralized mechanism for administering control of route imports between different VRFs.

**Note** BGP Accept Own is supported only for VPNv4 and VPNv6 address families in neighbor configuration mode.

Perform this task to configure BGP Accept Own:

**Procedure**

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)#router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)#neighbor 10.1.2.3
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)#remote-as 100
```

Assigns a remote autonomous system number to the neighbor.

**Step 5** **update-source** *type interface-path-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)#update-source Loopback0
```

Allows sessions to use the primary IP address from a specific interface as the local address when forming a session with a neighbor.

**Step 6** **address-family** {*vpnv4 unicast* | *vpnv6 unicast*}

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)#address-family vpnv6 unicast
```

Specifies the address family as VPNv4 or VPNv6 and enters neighbor address family configuration mode.

**Step 7** **accept-own** [**inheritance-disable**]

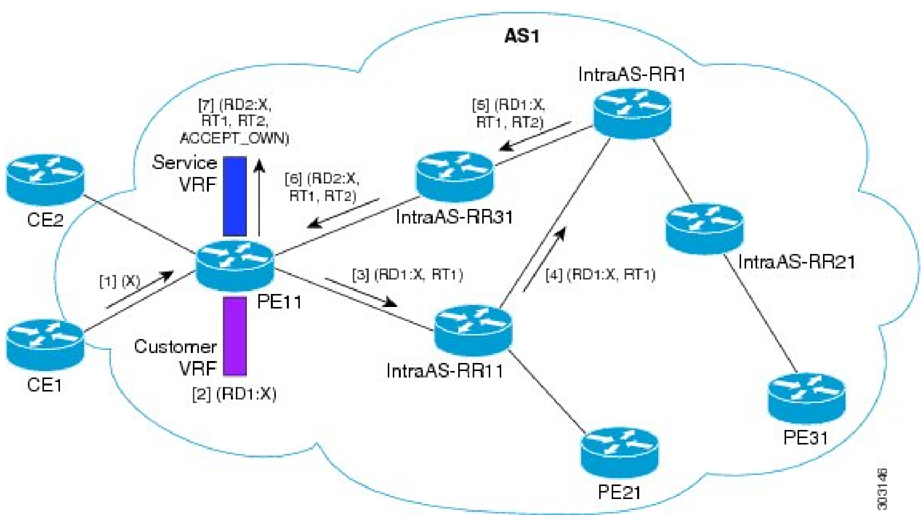
**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)#accept-own
```

Enables handling of self-originated VPN routes containing Accept_Own community.

Use the **inheritance-disable** keyword to disable the "accept own" configuration and to prevent inheritance of "acceptown" from a parent configuration.

## BGP Accept Own Configuration: Example




In this configuration example:

- PE11 is configured with Customer VRF and Service VRF.
- OSPF is used as the IGP.
- VPNv4 unicast and VPNv6 unicast address families are enabled between the PE and RR neighbors and IPv4 and IPv6 are enabled between PE and CE neighbors.

The Accept Own configuration works as follows:

1. CE1 originates prefix X.

2. Prefix X is installed in customer VRF as (RD1:X).
3. Prefix X is advertised to IntraAS-RR11 as (RD1:X, RT1).
4. IntraAS-RR11 advertises X to InterAS-RR1 as (RD1:X, RT1).
5. InterAS-RR1 attaches RT2 to prefix X on the inbound and ACCEPT_OWN community on the outbound and advertises prefix X to IntraAS-RR31.
6. IntraAS-RR31 advertises X to PE11.
7. PE11 installs X in Service VRF as (RD2:X,RT1, RT2, ACCEPT_OWN).

This example shows how to configure BGP Accept Own on a PE router.

```
router bgp 100
 neighbor 45.1.1.1
   remote-as 100
   update-source Loopback0
   address-family vpnv4 unicast
    route-policy pass-all in
    accept-own
    route-policy drop_111.x.x.x out
   !
   address-family vpnv6 unicast
    route-policy pass-all in
    accept-own
    route-policy drop_111.x.x.x out
   !
  !
```

This example shows an InterAS-RR configuration for BGP Accept Own.

```
router bgp 100
 neighbor 45.1.1.1
  remote-as 100
  update-source Loopback0
  address-family vpnv4 unicast
   route-policy rt_stitch1 in
   route-reflector-client
   route-policy add_bgp_ao out
  !
  address-family vpnv6 unicast
   route-policy rt_stitch1 in
   route-reflector-client
   route-policy add_bgp_ao out
  !
 !
extcommunity-set rt cs_100:1
  100:1
end-set
!
extcommunity-set rt cs_1001:1
  1001:1
end-set
!
route-policy rt_stitch1
  if extcommunity rt matches-any cs_100:1 then
    set extcommunity rt cs_1000:1 additive
 endif
end-policy
!
route-policy add_bgp_ao
  set community (accept-own) additive
```

```
end-policy
!
```

# BGP Link-State

BGP Link-State (LS) is an Address Family Identifier (AFI) and Sub-address Family Identifier (SAFI) defined to carry interior gateway protocol (IGP) link-state database through BGP. BGP LS delivers network topology information to topology servers and Application Layer Traffic Optimization (ALTO) servers. BGP LS allows policy-based control to aggregation, information-hiding, and abstraction. BGP LS supports IS-IS and OSPFv2.

**Note** IGPs do not use BGP LS data from remote peers. BGP does not download the received BGP LS data to any other component on the router.

## Configure BGP Link-state

To exchange BGP link-state (LS) information with a BGP neighbor, perform these steps:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family link-state link-state**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family link-state link-state
```

Distributes BGP link-state information to the specified neighbor.

**Step 4** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.0.0.2
```

Configures a CE neighbor. The ip-address argument must be a private address.

**Step 5** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 1
```

Configures the remote AS for the CE neighbor.

**Step 6** **address-family link-state link-state**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family link-state link-state
```

Distributes BGP link-state information to the specified neighbor.

**Step 7** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### Example

```
router bgp 100
 address-family link-state link-state
 !
 neighbor 10.0.0.2
  remote-as 1
  address-family link-state link-state
```

## Configure Domain Distinguisher

To configure unique identifier four-octet ASN, perform these steps:

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family link-state link-state**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family link-state link-state
```

Enters address-family link-state configuration mode.

**Step 4** **domain-distinguisher** *unique-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# domain-distinguisher 1234:1.2.3.4
```

Configures unique identifier four-octet ASN. Range is from 1 to 4294967295.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# BGP Permanent Network

BGP permanent network feature supports static routing through BGP. BGP routes to IPv4 or IPv6 destinations (identified by a route-policy) can be administratively created and selectively advertised to BGP peers. These routes remain in the routing table until they are administratively removed. A permanent network is used to define a set of prefixes as permanent, that is, there is only one BGP advertisement or withdrawal in upstream for a set of prefixes. For each network in the prefix-set, a BGP permanent path is created and treated as less preferred than the other BGP paths received from its peer. The BGP permanent path is downloaded into RIB when it is the best-path.

The **permanent-network** command in global address family configuration mode uses a route-policy to identify the set of prefixes (networks) for which permanent paths is to be configured. The **advertise permanent-network** command in neighbor address-family configuration mode is used to identify the peers to whom the permanent paths must be advertised. The permanent paths is always advertised to peers having the advertise permanent-network configuration, even if a different best-path is available. The permanent path is not advertised to peers that are not configured to receive permanent path.

The permanent network feature supports only prefixes in IPv4 unicast and IPv6 unicast address-families under the default Virtual Routing and Forwarding (VRF).

### Restrictions

These restrictions apply while configuring the permanent network:

- Permanent network prefixes must be specified by the route-policy on the global address family.
- You must configure the permanent network with route-policy in global address family configuration mode and then configure it on the neighbor address family configuration mode.
- When removing the permanent network configuration, remove the configuration in the neighbor address family configuration mode and then remove it from the global address family configuration mode.

## Configure BGP Permanent Network

Perform this task to configure BGP permanent network. You must configure at least one route-policy to identify the set of prefixes (networks) for which the permanent network (path) is to be configured.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **prefix-set** *prefix-set-name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# prefix-set PERMANENT-NETWORK-IPv4
RP/0/RP0/CPU0:router(config-pfx)# 1.1.1.1/32,
RP/0/RP0/CPU0:router(config-pfx)# 2.2.2.2/32,
RP/0/RP0/CPU0:router(config-pfx)# 3.3.3.3/32
RP/0/RP0/CPU0:router(config-pfx)# end-set
```

Enters prefix set configuration mode and defines a prefix set for contiguous and non-contiguous set of bits.

**Step 3** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-pfx)# exit
```

Exits prefix set configuration mode and enters global configuration mode.

**Step 4** **route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# route-policy POLICY-PERMANENT-NETWORK-IPv4
RP/0/RP0/CPU0:router(config-rpl)# if destination in PERMANENT-NETWORK-IPv4 then
RP/0/RP0/CPU0:router(config-rpl)# pass
RP/0/RP0/CPU0:router(config-rpl)# endif
```

Creates a route policy and enters route policy configuration mode, where you can define the route policy.

**Step 5** **end-policy**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# end-policy
```

Ends the definition of a route policy and exits route policy configuration mode.

**Step 6** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode.

**Step 7** **address-family { ipv4 | ipv6 } unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

**Step 8** **permanent-network route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# permanent-network route-policy
POLICY-PERMANENT-NETWORK-IPv4
```

Configures the permanent network (path) for the set of prefixes as defined in the route-policy.

**Step 9** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

**Step 10** **show bgp {ipv4 | ipv6} unicast** *prefix-set*

**Example:**

```
RP/0/RP0/CPU0:routershow bgp ipv4 unicast
```

(Optional) Displays whether the prefix-set is a permanent network in BGP.

## Advertise Permanent Network

Perform this task to identify the peers to whom the permanent paths must be advertised.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.255.255.254
```

Places the router in neighbor configuration mode for BGP routing and configures the neighbor IP address as a BGP peer.

**Step 4** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 4713
```

Assigns the neighbor a remote autonomous system number.

**Step 5** **address-family { ipv4 | ipv6 } unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family ipv4 unicast
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

**Step 6** **advertise permanent-network**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# advertise permanent-network
```

Specifies the peers to whom the permanent network (path) is advertised.

**Step 7** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

**Step 8** **show bgp {ipv4 | ipv6} unicast neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:routershow bgp ipv4 unicast neighbor 10.255.255.254
```

(Optional) Displays whether the neighbor is capable of receiving BGP permanent networks.

# Enable BGP Unequal Cost Recursive Load Balancing

### Procedure

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **configure**<br>**Example:**<br>`RP/0/RP0/CPU0:router# configure` | Enters mode. |
| **Step 2** | **router bgp** *as-number*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config)# router bgp 120` | Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process. |
| **Step 3** | **address-family** { **ipv4** \| **ipv6** } **unicast**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast` | Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.<br>To see a list of all the possible keywords and arguments for this command, use the CLI help (?). |

| | Command or Action | Purpose |
|---|---|---|
| **Step 4** | **maximum-paths** { **ebgp** \| **ibgp** \| **eibgp** } *maximum* [ **unequal-cost** ]<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-af)# maximum-paths ebgp 3` | Configures the maximum number of parallel routes that BGP installs in the routing table.<br>• **ebgp** *maximum* : Consider only eBGP paths for multipath.<br>• **ibgp** *maximum* [ **unequal-cost** ]: Consider load balancing between iBGP learned paths.<br>• **eibgp** *maximum* : Consider both eBGP and iBGP learned paths for load balancing. eiBGP load balancing always does unequal-cost load balancing.<br>When eiBGP is applied, eBGP or iBGP load balancing cannot be configured; however, eBGP and iBGP load balancing can coexist. |
| **Step 5** | **exit**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-af)# exit` | Exits the current configuration mode. |
| **Step 6** | **neighbor** *ip-address*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.0.0.0` | Configures a CE neighbor. The *ip-address* argument must be a private address. |
| **Step 7** | **dmz-link-bandwidth**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-nbr)# dmz-link-bandwidth` | Originates a demilitarized-zone (DMZ) link-bandwidth extended community for the link to an eBGP/iBGP neighbor. |
| **Step 8** | Use the **commit** or **end** command. | **commit** —Saves the configuration changes and remains within the configuration session.<br>**end** —Prompts user to take one of these actions:<br>• **Yes** — Saves configuration changes and exits the configuration session.<br>• **No** —Exits the configuration session without committing the configuration changes.<br>• **Cancel** —Remains in the configuration session, without committing the configuration changes. |

# DMZ Link Bandwidth for Unequal Cost Recursive Load Balancing

The demilitarized zone (DMZ) link bandwidth for unequal cost recursive load balancing feature provides support for unequal cost load balancing for recursive prefixes on local node using DMZ link bandwidth. Use the dmz-link-bandwidth command in BGP neighbor configuration mode and the bandwidth command in interface configuration mode to The unequal load balance is achieved.

When the PE router includes the link bandwidth extended community in its updates to the remote PE through the Multiprotocol Interior BGP (MP-iBGP) session (either IPv4 or VPNv4), the remote PE automatically does load balancing if the **maximum-paths** command is enabled.

**Note** Unequal cost recursive load balancing happens across maximum eight paths only.

## Enable BGP Unequal Cost Recursive Load Balancing

### Procedure

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **configure**<br>**Example:**<br>`RP/0/RP0/CPU0:router# configure` | Enters mode. |
| **Step 2** | **router bgp** *as-number*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config)# router bgp 120` | Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process. |
| **Step 3** | **address-family** { **ipv4** \| **ipv6** } **unicast**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast` | Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.<br>To see a list of all the possible keywords and arguments for this command, use the CLI help (?). |
| **Step 4** | **maximum-paths** { **ebgp** \| **ibgp** \| **eibgp** } *maximum* [ **unequal-cost** ]<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-af)# maximum-paths ebgp 3` | Configures the maximum number of parallel routes that BGP installs in the routing table.<br>• **ebgp** *maximum* : Consider only eBGP paths for multipath.<br>• **ibgp** *maximum* [ **unequal-cost** ]: Consider load balancing between iBGP learned paths.<br>• **eibgp** *maximum* : Consider both eBGP and iBGP learned paths for load balancing. eiBGP load balancing always does unequal-cost load balancing. |

| | Command or Action | Purpose |
|---|---|---|
| | | When eiBGP is applied, eBGP or iBGP load balancing cannot be configured; however, eBGP and iBGP load balancing can coexist. |
| **Step 5** | **exit**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-af)# exit` | Exits the current configuration mode. |
| **Step 6** | **neighbor** *ip-address*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.0.0.0` | Configures a CE neighbor. The *ip-address* argument must be a private address. |
| **Step 7** | **dmz-link-bandwidth**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-nbr)# dmz-link-bandwidth` | Originates a demilitarized-zone (DMZ) link-bandwidth extended community for the link to an eBGP/iBGP neighbor. |
| **Step 8** | Use the **commit** or **end** command. | **commit** —Saves the configuration changes and remains within the configuration session.<br>**end** —Prompts user to take one of these actions:<br>• **Yes** — Saves configuration changes and exits the configuration session.<br>• **No** —Exits the configuration session without committing the configuration changes.<br>• **Cancel** —Remains in the configuration session, without committing the configuration changes. |

## DMZ Link Bandwidth Over EBGP Peer

The demilitarized zone (DMZ) link bandwidth extended community is an optional non-transitive attribute; therefore, it is not advertised to eBGP peers by default but it is advertised only to iBGP peers. This extended community is meant for load balancing over multi-paths. However, Cisco IOS-XR enables advertising of the DMZ link bandwidth to an eBGP peer, or receiving the DMZ link bandwidth by an eBGP peer. This feature also gives the user the option to send the bandwidth unchanged, or take the accumulated bandwidth over all the egress links and advertise that to the upstream eBGP peer.

Use the **ebgp-send-community-dmz** command to send the community to eBGP peers. By default, the link bandwidth extended-community attribute associated with the best path is sent.

When the **cumulative** keyword is used, the value of the link bandwidth extended community is set to the sum of link bandwidth values of all the egress-multipaths. If the DMZ link bandwidth value of the multipaths is unknown, for instance, some paths do not have that attribute, then unequal cost load-balancing is not done at that node. However, the sum of the known DMZ link bandwidth values is calculated and sent to the eBGP peer.

Use the **ebgp-recv-community-dmz** command to receive the community from eBGP peers.

> **Note** The **ebgp-send-community-dmz** and **ebgp-recv-community-dmz** commands can be configured in the neigbor, neighbour-group, and session-group configuration mode.

Use the **bgp bestpath as-path multipath-relax**and **bgp bestpath as-path ignore** commands to handle multipath across different autonomous systems.

## Sending and Receiving DMZ Link Bandwidth Extended Community over eBGP Peer

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.1.1.1
```

Enters the neighbor configuration mode for configuring BGP routing sessions.

**Step 4** **ebgp-send-extcommunity-dmz** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# ebgp-send-extcommunity-dmz
```

Sends the DMZ link bandwidth extended community to the eBGP neighbor.

> **Note** Use the **cumulative** keyword with this command to set the value of the link bandwidth extended community to the sum of link bandwidth values of all the egress multipaths.

**Step 5** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# exit
```

Exits the neighbor configuration mode and enters into BGP configuration mode.

**Step 6** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.16.0.1
```

Enters the neighbor configuration mode for configuring BGP routing sessions.

**Step 7** **ebgp-recv-extcommunity-dmz**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# ebgp-recv-extcommunity-dmz
```

Receives the DMZ link bandwidth extended community to the eBGP neighbor.

**Step 8** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# exit
```

Exits the neighbor configuration mode and enters into BGP configuration mode.

### DMZ Link Bandwidth: Example

The following examples shows how Router R1 sends DMZ link bandwidth extended communities to Router R2 over eBGP peer connection:

```
R1: sending router
------------------
neighbour 10.3.3.3
  remote-as 2
  ebgp-send-extcommunity-dmz
  address-family ipv4 unicast
   route-policy pass in
   route-policy pass out
  !

R2: Receiving router
--------------------
neighbor 192.0.2.1
  remote-as 3
  ebgp-recv-extcommunity-dmz
  address-family ipv4 unicast
   route-policy pass in
  !
route-policy pass out
!
```

The following is a sample configuration that displays the DMZ link bandwidth configuration in the sending (R1) router:

```
RP/0/RP0/CPU0:router)# show bgp ipv4 unicast 10.1.1.1/32 detail

Path #1: Received by speaker 0
  Flags: 0x4000000001040003, import: 0x20
  Advertised to update-groups (with more than one peer):
    0.4
  Advertised to peers (in unique update groups):
    20.0.0.1
  3
    11.1.0.2 from 11.1.0.2 (11.1.0.2)
      Origin incomplete, metric 20, localpref 100, valid, external, best, group-best
      Received Path ID 0, Local Path ID 0, version 21
      Extended community: LB:3:192
      Origin-AS validity: not-found
```

The following is a sample configuration that displays DMZ link bandwidth configuration in the receiving (R2) router:

```
RP/0/RP0/CPU0:router)# show bgp ipv4 unicast 10.1.1.1/32 detail

Paths: (1 available, best #1)
  Not advertised to any peer
  Path #1: Received by speaker 0
  Not advertised to any peer
  1 3
    20.0.0.2 from 20.0.0.2 (10.0.0.81)
      Origin incomplete, localpref 100, valid, external, best, group-best
      Received Path ID 0, Local Path ID 0, version 17
      Extended community: LB:1:192
      Origin-AS validity: not-found
```

# BGP Prefix Origin Validation using RPKI

A BGP route associates an address prefix with a set of autonomous systems (AS) that identify the interdomain path the prefix has traversed in the form of BGP announcements. This set is represented as the AS_PATH attribute in BGP and starts with the AS that originated the prefix.

To help reduce well-known threats against BGP including prefix mis-announcing and monkey-in-the-middle attacks, one of the security requirements is the ability to validate the origination AS of BGP routes. The AS number claiming to originate an address prefix (as derived from the AS_PATH attribute of the BGP route) needs to be verified and authorized by the prefix holder.

The Resource Public Key Infrastructure (RPKI) is an approach to build a formally verifiable database of IP addresses and AS numbers as resources. The RPKI is a globally distributed database containing, among other things, information mapping BGP (internet) prefixes to their authorized origin-AS numbers. Routers running BGP can connect to the RPKI to validate the origin-AS of BGP paths.

## Configure RPKI Cache-server

Perform this task to configure Resource Public Key Infrastructure (RPKI) cache-server parameters.

Configure the RPKI cache-server parameters in rpki-server configuration mode. Use the **rpki server** command in router BGP configuration mode to enter into the rpki-server configuration mode

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)#router bgp 100
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **rpki cache** {*host-name* | *ip-address*}

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)#rpki server 10.2.3.4
```

Enters rpki-server configuration mode and enables configuration of RPKI cache parameters.

**Step 4** Use one of these commands:

- **transport ssh port** *port_number*
- **transport tcp port** *port_number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#transport ssh port 22
```

Or

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#transport tcp port 2
```

Specifies a transport method for the RPKI cache.

- **ssh**—Select **ssh** to connect to the RPKI cache using SSH.
- **tcp**—Select **tcp** to connect to the RPKI cache using TCP (unencrypted).
- **port** *port_number*—Specify a port number for the specified RPKI cache transport. For tcp, the range of supported port number is 1 to 65535. For ssh, use port number 22.

**Note**
- Do not specify a custom port number for RPKI cache transport over SSH. You must use port 22 for RPKI over SSH.
- You can set the transport to either TCP or SSH. Change of transport causes the cache session to flap.

**Step 5** (Optional) **username** *user_name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#username ssh_rpki_cache
```

Specifies a (SSH) username for the RPKI cache-server.

**Step 6** (Optional) **password**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#password ssh_rpki_pass
```

Specifies a (SSH) password for the RPKI cache-server.

**Note** The “username” and “password” configurations only apply if the SSH method of transport is active.

**Step 7** **preference** *preference_value*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#preference 1
```

Specifies a preference value for the RPKI cache. Range for the preference value is 1 to 10. Setting a lower preference value is better.

**Step 8** **purge-time** *time*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#purge-time 30
```

Configures the time BGP waits to keep routes from a cache after the cache session drops. Set purge time in seconds. Range for the purge time is 30 to 360 seconds.

**Step 9** Use one of these commands.

- **refresh-time** *time*
- **refresh-time off**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#refresh-time 20
```

Or

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#refresh-time off
```

Configures the time BGP waits in between sending periodic serial queries to the cache. Set refresh-time in seconds. Range for the refresh time is 15 to 3600 seconds.

Configure the **off** option to specify not to send serial-queries periodically.

**Step 10** Use one these commands.

- **response-time** *time*
- **response-time off**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#response-time 30
```

Or

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#response-time off
```

Configures the time BGP waits for a response after sending a serial or reset query. Set response-time in seconds. Range for the response time is 15 to 3600 seconds.

Configure the **off** option to wait indefinitely for a response.

**Step 11** **shutdown**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-rpki-server)#shutdown
```

Configures shut down of the RPKI cache.

**Step 12** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Configure BGP Prefix Validation

Starting from Release 6.5.1, RPKI is disabled by default. From Release 6.5.1, use the following task to configure RPKI Prefix Validation.

```
Router(config)# router bgp 100
/* The bgp origin-as validation time and bgp origin-as validity signal ibgp commands are
optional. */.
Router(config-bgp)# bgp origin-as validation time 50
Router(config-bgp)# bgp origin-as validation time off
Router(config-bgp)# bgp origin-as validation signal ibgp
Router(config-bgp)# address-family ipv4 unicast
Router(config-bgp-af)# bgp origin-as validation enable
```

Use the following commands to verify the origin-as validation configuration:

```
Router# show bgp origin-as validity

Thu Mar 14 04:18:09.656 PDT
BGP router identifier 10.1.1.1, local AS number 1
BGP generic scan interval 60 secs
Non-stop routing is enabled
BGP table state: Active
Table ID: 0xe0000000   RD version: 514
BGP main routing table version 514
BGP NSR Initial initsync version 2 (Reached)
BGP NSR/ISSU Sync-Group versions 0/0
BGP scan interval 60 secs
Status codes: s suppressed, d damped, h history, * valid, > best
              i - internal, r RIB-failure, S stale, N Nexthop-discard
Origin codes: i - IGP, e - EGP, ? - incomplete
Origin-AS validation codes: V valid, I invalid, N not-found, D disabled
    Network               Next Hop            Metric LocPrf Weight Path
 *> 209.165.200.223/27    0.0.0.0                  0         32768 ?

 *> 209.165.200.225/27    0.0.0.0                  0         32768 ?

 *> 19.1.2.0/24           0.0.0.0                  0         32768 ?

 *> 19.1.3.0/24           0.0.0.0                  0         32768 ?

 *> 10.1.2.0/24           0.0.0.0                  0         32768 ?
```

```
 *> 10.1.3.0/24            0.0.0.0                  0         32768 ?

 *> 10.1.4.0/24            0.0.0.0                  0         32768 ?

 *> 198.51.100.1/24        0.0.0.0                  0         32768 ?

 *> 203.0.113.235/24       0.0.0.0                  0         32768 ?

V*> 209.165.201.0/27      10.1.2.1                  0         4002 i

N*> 198.51.100.2/24       10.1.2.1                  0         4002 i

I*> 198.51.100.1/24       10.1.2.1                  0          4002 i

 *> 192.0.2.1.0/24        0.0.0.0                    0         32768 ?
Router# show bgp process
Mon Jul  9 16:47:39.428 PDT

BGP Process Information:
...
Use origin-AS validity in bestpath decisions
Allow (origin-AS) INVALID paths
Signal origin-AS validity state to neighbors

Address family: IPv4 Unicast
...
Origin-AS validation is enabled for this address-family
Use origin-AS validity in bestpath decisions for this address-family
Allow (origin-AS) INVALID paths for this address-family
Signal origin-AS validity state to neighbors with this address-family
```

## Configure RPKI Bestpath Computation

Perform this task to configure RPKI bestpath computation options.

### Procedure

---

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)#router bgp 100
```

Specifies the BGP AS number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **rpki bestpath use origin-as validity**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)#bgp bestpath origin-as use validity
```

Enables the validity states of BGP paths to affect the path's preference in the BGP bestpath process. This configuration can also be done in router BGP address family submode.

**Step 4** **rpki bestpath origin-as allow invalid**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)#bgp bestpath origin-as allow invalid
```

Allows all "invalid" paths to be considered for BGP bestpath computation.

**Note** This configuration can also be done at global address family, neighbor, and neighbor address family submodes. Configuring rpki bestpath origin-as allow invalid in router BGP and address family submodes allow all "invalid" paths to be considered for BGP bestpath computation. By default, all such paths are not bestpath candidates. Configuring pki bestpath origin-as allow invalid in neighbor and neighbor address family submodes allow all "invalid" paths from that specific neighbor or neighbor address family to be considered as bestpath candidates. The neighbor must be an eBGP neighbor.

This configuration takes effect only when the **rpki bestpath use origin-as validity** configuration is enabled.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

# Resilient Per-CE Label Allocation Mode

The Resilient Per-CE Label Allocation is an extension of the Per-CE label allocation mode to support Prefix Independent Convergence (PIC) and load balancing. At present, the three label allocation modes, Per-Prefix, Per-CE, and Per-VRF have these restrictions:

- No support for load balancing across CEs
- Temporary forwarding loop during local traffic diversion to support PIC
- No support for EIBGP multipath load balancing
- Forwarding performance impact

In the Resilient Per-CE label allocation scheme, BGP installs a unique rewrite label in LSD for every unique set of CE paths or next hops. There may be one or more prefixes in BGP table that points to this label. BGP also installs the CE paths (primary) and optionally a backup PE path into RIB. FIB learns about the label rewrite information from LSD and the IP paths from RIB. In steady state, labeled traffic destined to the resilient per-CE label is load balanced across all the CE next hops. When all the CE paths fail, any traffic destined to that label will result in an IP lookup and will be forwarded towards the backup PE path, if available. This action is performed on the label independently of the number of prefixes that may point to the label, resulting in the PIC behavior during primary paths failure.

## Configure Resilient Per-CE Label Allocation Mode Under VRF Address Family

Perform this task to configure resilient per-ce label allocation mode under VRF address family.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)#
```

Enters global configuration mode.

**Step 2** **router bgp***as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 666
RP/0/RP0/CPU0:router(config-bgp)#
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **vrf***vrf-instance*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# vrf vrf-pe
RP/0/RP0/CPU0:router(config-bgp-vrf)#
```

Configures a VRF instance.

**Step 4** **address-family {ipv4 | ipv6} unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-bgp-vrf-af)#
```

Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.

**Step 5** **label-mode per-ce**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# label mode per-ce
RP/0/RP0/CPU0:router(config-bgp-vrf-af)#
```

Configures resilient per-ce label allocation mode.

**Step 6** Do one of the following:

- **end**
- **commit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# end
```

or

```
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# commit
```

Saves configuration changes.

- When you issue the **end** command, the system prompts you to commit changes:

  ```
  Uncommitted changes found, commit them before exiting(yes/no/cancel)?[cancel]:
  ```

  - Entering **yes** saves configuration changes to the running configuration file, exits the configuration session, and returns the router to EXEC mode.
  - Entering **no** exits the configuration session and returns the router to EXEC mode without committing the configuration changes.
  - Entering **cancel** leaves the router in the current configuration session without exiting or committing the configuration changes.

- Use the **commit** command to save the configuration changes to the running configuration file and remain within the configuration session.

This example shows how to configure resilient per-ce label allocation mode under VRF address family:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# router bgp 666
RP/0/RP0/CPU0:router(config-bgp)# vrf vrf-pe
RP/0/RP0/CPU0:router(config-bgp-vrf)# address-family ipv4 unicast
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# label mode per-ce
RP/0/RP0/CPU0:router(config-bgp-vrf-af)# end
```

## Configure Resilient Per-CE Label Allocation Mode Using Route-Policy

Perform this task to configure resilient per-ce label allocation mode using a route-policy.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)#
```

Enters global configuration mode.

**Step 2** **route-policy***policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# route-policy route1
RP/0/RP0/CPU0:router(config-rpl)#
```

Creates a route policy and enters route policy configuration mode.

**Step 3** **set label-mode per-ce**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# set label-mode per-ce
RP/0/RP0/CPU0:router(config-rpl)#
```

Configures resilient per-ce label allocation mode.

**Step 4** Do one of the following:

- **end**
- **commit**

**Example:**

```
RP/0/RP0/CPU0:router(config-rpl)# end
```

or

```
RP/0/RP0/CPU0:router(config-rpl)# commit
```

Saves configuration changes.

- When you issue the **end** command, the system prompts you to commit changes:

  ```
  Uncommitted changes found, commit them before exiting(yes/no/cancel)?[cancel]:
  ```

  - Entering **yes** saves configuration changes to the running configuration file, exits the configuration session, and returns the router to EXEC mode.
  - Entering **no** exits the configuration session and returns the router to EXEC mode without committing the configuration changes.
  - Entering **cancel** leaves the router in the current configuration session without exiting or committing the configuration changes.

- Use the **commit** command to save the configuration changes to the running configuration file and remain within the configuration session.

---

This example shows how to configure resilient per-ce label allocation mode using a route-policy:

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# route-policy route1
RP/0/RP0/CPU0:router(config-rpl)# set label-mode per-ce
RP/0/RP0/CPU0:router(config-rpl)# end
```

This example shows how to configure route-policy to vrf :

```
RP/0/RP0/CPU0:(config)# router bgp 666
RP/0/RP0/CPU0:(config-bgp)# vrf vrf-pe
RP/0/RP0/CPU0:(config-bgp-vrf)# address-family ipv4 unicast
RP/0/RP0/CPU0:(config-bgp-vrf-af)# label mode route-policy route1
```

# BGP VRF Dynamic Route Leaking

The Border Gateway Protocol (BGP) dynamic route leaking feature provides the ability to import routes between the default-vrf (Global VRF) and any other non-default VRF, to provide connectivity between a global and a VPN host. The import process installs the Internet route in a VRF table or a VRF route in the Internet table, providing connectivity.

The dynamic route leaking is enabled by:

- Importing from default-VRF to non-default-VRF, using the **import from default-vrf route-policy** *route-policy-name* [**advertise-as-vpn**] command in VRF address-family configuration mode.

  If the **advertise-as-vpn** option is configured, the paths imported from the default-VRF to the non-default-VRF are advertised to the PEs as well as to the CEs. If the **advertise-as-vpn** option is not configured, the paths imported from the default-VRF to the non-default-VRF are not advertised to the PE. However, the paths are still advertised to the CEs.

- Importing from non-default-VRF to default VRF, using the **export to default-vrf route-policy** *route-policy-name* command in VRF address-family configuration mode.

A route-policy is mandatory to filter the imported routes. This reduces the risk of unintended import of routes between the Internet table and the VRF tables and the corresponding security issues. There is no hard limit on the number of prefixes that can be imported. The import creates a new prefix in the destination VRF, which increases the total number of prefixes and paths. However, each VRF importing global routes adds workload equivalent to a neighbor receiving the global table. This is true even if the user filters out all but a few prefixes. Hence, importing five to ten VRFs is ideal.

## Configure VRF Dynamic Route Leaking

Perform these steps to import routes from default-VRF to non-default VRF or to import routes from non-default VRF to default VRF.

### Before you begin

A route-policy is mandatory for configuring dynamic route leaking. Use the **route-policy** *route-policy-name* command in global configuration mode to configure a route-policy.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **vrf** *vrf_name*

**Example:**

```
RP/0/RSP0/CPU0:PE51_ASR-9010(config)#vrf vrf_1
```

Enters VRF configuration mode.

**Step 3** **address-family** {**ipv4** | **ipv6**} **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf)#address-family ipv6 unicast
```

Enters VRF address-family configuration mode.

**Step 4** Use one of these options:

- **import from default-vrf route-policy** *route-policy-name* [**advertise-as-vpn**]
- **export to default-vrf route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf-af)#import from default-vrf route-policy
rpl_dynamic_route_import
```

or

```
RP/0/RP0/CPU0:router(config-vrf-af)#export to default-vrf route-policy
rpl_dynamic_route_export
```

Imports routes from default-VRF to non-default VRF or from non-default VRF to default-VRF.

- **import from default-vrf**—configures import from default-VRF to non-default-VRF.

  If the **advertise-as-vpn** option is configured, the paths imported from the default-VRF to the non-default-VRF are advertised to the PEs as well as to the CEs. If the **advertise-as-vpn** option is not configured, the paths imported from the default-VRF to the non-default-VRF are not advertised to the PE. However, the paths are still advertised to the CEs.

- **export to default-vrf**—configures import from non-default-VRF to default VRF. The paths imported from the default-VRF are advertised to other PEs.

**Step 5** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

### VRF Dynamic Route Leaking Configuration: Example

Import Routes from default-VRF to non-default-VRF:

```
vrf vrf_1
 address-family ipv6 unicast
  import from default-vrf route-policy rpl_dynamic_route_import
```

```
 !
end
```

Import Routes from non-default-VRF to default-VRF

```
vrf vrf_1
 address-family ipv6 unicast
    export to default-vrf route-policy rpl_dynamic_route_export
 !
end
```

#### What to do next

These **show bgp** command output displays information from the dynamic route leaking configuration:

- Use the **show bgp prefix** command to display the source-RD and the source-VRF for imported paths, including the cases when IPv4 or IPv6 unicast prefixes have imported paths.
- Use the **show bgp imported-routes** command to display IPv4 unicast and IPv6 unicast address-families under the default-VRF.

# Configuring a VPN Routing and Forwarding Instance in BGP

Layer 3 (virtual private network) VPN can be configured only if there is an available Layer 3 VPN license for the line card slot on which the feature is being configured. If advanced IP license is enabled, 4096 Layer 3 VPN routing and forwarding instances (VRFs) can be configured on an interface. If the infrastructure VRF license is enabled, eight Layer 3 VRFs can be configured on the line card.

The following error message appears if the appropriate licence is not enabled:

```
RP/0/RP0/CPU0:router#LC/0/0/CPU0:Dec 15 17:57:53.653 : rsi_agent[247]:
%LICENSE-ASR9K_LICENSE-2-INFRA_VRF_NEEDED : 5 VRF(s) are configured without license
A9K-iVRF-LIC in violation of the Software Right To Use Agreement.
This feature may be disabled by the system without the appropriate license.
Contact Cisco to purchase the license immediately to avoid potential service interruption.
```

**Note** An AIP license is not required for configuring L2VPN services.

The following tasks are used to configure a VPN routing and forwarding (VRF) instance in BGP:

## Define Virtual Routing and Forwarding Tables in Provider Edge Routers

Perform this task to define the VPN routing and forwarding (VRF) tables in the provider edge (PE) routers.

#### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **vrf** *vrf-name*

**Example:**

```
RP/0/RP0/CPU0:router(config)# vrf vrf_pe
```

Configures a VRF instance.

**Step 3** **address-family** { **ipv4** | **ipv6** } **unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf)# address-family ipv4 unicast
```

Specifies either the IPv4 or IPv6 address family and enters address family configuration submode.

To see a list of all the possible keywords and arguments for this command, use the CLI help (?).

**Step 4** **maximum prefix** *maximum* [ *threshold* ]

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf-af)# maximum prefix 2300
```

Configures a limit to the number of prefixes allowed in a VRF table.

A maximum number of routes is applicable to dynamic routing protocols as well as static or connected routes.

You can specify a threshold percentage of the prefix limit using the *mid-threshold* argument.

**Step 5** **import route-policy** *policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf-af)# import route-policy policy_a
```

(Optional) Provides finer control over what gets imported into a VRF. This import filter discards prefixes that do not match the specified *policy-name* argument.

**Step 6** **import route-target** [ *as-number* **:** *nn* | *ip-address* **:** *nn* ]

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf-af)# import route-target 234:222
```

Specifies a list of route target (RT) extended communities. Only prefixes that are associated with the specified import route target extended communities are imported into the VRF.

**Step 7** **export route-policy** *policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-vrf-af)# export route-policy policy_b
```

(Optional) Provides finer control over what gets exported into a VRF. This export filter discards prefixes that do not match the specified *policy-name* argument.

**Step 8** **export route-target** [ *as-number* **:** *nn* | *ip-address* **:** *nn* ]

**Example:**

```
RP/0/RP0/CPU0:routerr(config-vrf-af)# export route-target 123:234
```

Specifies a list of route target extended communities. Export route target communities are associated with prefixes when they are advertised to remote PEs. The remote PEs import them into VRFs which have import RTs that match these exported route target communities.

**Step 9** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Configure Route Distinguisher

The route distinguisher (RD) makes prefixes unique across multiple VPN routing and forwarding (VRF) instances.

In the L3VPN multipath same route distinguisher (RD)environment, the determination of whether to install a prefix in RIB or not is based on the prefix's bestpath. In a rare misconfiguration situation, where the best pah is not a valid path to be installed in RIB, BGP drops the prefix and does not consider the other paths. The behavior is different for different RD setup, where the non-best multipath will be installed if the best multipath is invalid to be installed in RIB.

Perform this task to configure the RD.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Enters BGP configuration mode allowing you to configure the BGP routing process.

**Step 3** **bgp router-id** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# bgp router-id 10.0.0.0
```

Configures a fixed router ID for the BGP-speaking router.

**Step 4** **vrf** *vrf-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# vrf vrf_pe
```

Configures a VRF instance.

**Step 5** **rd** { *as-number* **:** *nn* | *ip-address* **:** *nn* | **auto** }

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf)# rd 345:567
```

Configures the route distinguisher.

Use the **auto** keyword if you want the router to automatically assign a unique RD to the VRF.

Automatic assignment of RDs is possible only if a router ID is configured using the **bgp router-id** command in router configuration mode. This allows you to configure a globally unique router ID that can be used for automatic RD generation. The router ID for the VRF does not need to be globally unique, and using the VRF router ID would be incorrect for automatic RD generation. Having a single router ID also helps in checkpointing RD information for BGP graceful restart, because it is expected to be stable across reboots.

**Step 6** Do one of the following:

- **end**
- **commit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-vrf)# end
```

or

```
RP/0/RP0/CPU0:router(config-bgp-vrf)# commit
```

Saves configuration changes.

- When you issue the **end** command, the system prompts you to commit changes:

```
Uncommitted changes found, commit them before exiting(yes/no/cancel)?[cancel]:
```

  - Entering **yes** saves configuration changes to the running configuration file, exits the configuration session, and returns the router to XR EXEC mode.
  - Entering **no** exits the configuration session and returns the router to XR EXEC mode without committing the configuration changes.
  - Entering **cancel** leaves the router in the current configuration session without exiting or committing the configuration changes.

- Use the **commit** command to save the configuration changes to the running configuration file and remain within the configuration session.

## Configure PE-PE or PE-RR Interior BGP Sessions

To enable BGP to carry VPN reachability information between provider edge (PE) routers you must configure the PE-PE interior BGP (iBGP) sessions. A PE uses VPN information carried from the remote PE router to determine VPN connectivity and the label value to be used so the remote (egress) router can demultiplex the packet to the correct VPN during packet forwarding.

The PE-PE, PE-route reflector (RR) iBGP sessions are defined to all PE and RR routers that participate in the VPNs configured in the PE router.

Perform this task to configure PE-PE iBGP sessions and to configure global VPN options on a PE.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters mode.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 120
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family vpnv4 unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family vpvn4 unicast
```

Enters VPN address family configuration mode.

**Step 4** **exit**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# exit
```

Exits the current configuration mode.

**Step 5** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 172.16.1.1
```

Configures a PE iBGP neighbor.

**Step 6** **remote-as** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# remote-as 1
```

Assigns the neighbor a remote autonomous system number.

**Step 7** **description** *text*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# description neighbor 172.16.1.1
```

(Optional) Provides a description of the neighbor. The description is used to save comments and does not affect software function.

**Step 8** **password** { **clear** | **encrypted** } *password*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# password encrypted 123abc
```

Enables Message Digest 5 (MD5) authentication on the TCP connection between the two BGP neighbors.

**Step 9** **shutdown**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# shutdown
```

Terminates any active sessions for the specified neighbor and removes all associated routing information.

**Step 10** **timers** *keepalive hold-time*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# timers 12000 200
```

Set the timers for the BGP neighbor.

**Step 11** **update-source** *type interface-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# update-source gigabitEthernet 0/1/5/0
```

Allows iBGP sessions to use the primary IP address from a specific interface as the local address when forming an iBGP session with a neighbor.

**Step 12** **address-family** **vpnv4 unicast**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family vpnv4 unicast
```

Enters VPN neighbor address family configuration mode.

**Step 13** **route-policy** *route-policy-name* **in**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy pe-pe-vpn-in in
```

Specifies a routing policy for an inbound route. The policy can be used to filter routes or modify route attributes.

**Step 14** **route-policy** *route-policy-name* **out**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr-af)# route-policy pe-pe-vpn-out out
```

Specifies a routing policy for an outbound route. The policy can be used to filter routes or modify route attributes.

**Step 15** Use the **commit** or **end** command.

**commit** —Saves the configuration changes and remains within the configuration session.

**end** —Prompts user to take one of these actions:

- **Yes** — Saves configuration changes and exits the configuration session.
- **No** —Exits the configuration session without committing the configuration changes.
- **Cancel** —Remains in the configuration session, without committing the configuration changes.

## Configure BGP as PE-CE Protocol

Perform this task to configure BGP on the PE and establish PE-CE communication using BGP.

### Procedure

| | Command or Action | Purpose |
|---|---|---|
| **Step 1** | **configure**<br>**Example:**<br>`RP/0/RP0/CPU0:router# configure` | Enters mode. |
| **Step 2** | **router bgp** *as-number*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config)# router bgp 120` | Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process. |
| **Step 3** | **vrf** *vrf-name*<br>**Example:** | Enables BGP routing for a particular VRF on the PE router. |

| | Command or Action | Purpose |
|---|---|---|
| | `RP/0/RP0/CPU0:router(config-bgp)# vrf vrf_pe_2` | |
| Step 4 | **bgp router-id** *ip-address*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf)# bgp router-id 172.16.9.9` | Configures a fixed router ID for a BGP-speaking router. |
| Step 5 | **label-allocation-mode per-ce**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf)# label-allocation-mode per-ce` | • Configures The **per-ce** keyword configures the per-CE label allocation mode to avoid an extra lookup on the PE router and conserve label space (per-prefix is the default label allocation mode). In this mode, the PE router allocates one label for every immediate next-hop (in most cases, this would be a CE router). This label is directly mapped to the next hop, so there is no VRF route lookup performed during data forwarding. However, the number of labels allocated would be one for each CE rather than one for each VRF. Because BGP knows all the next hops, it assigns a label for each next hop (not for each PE-CE interface). When the outgoing interface is a multiaccess interface and the media access control (MAC) address of the neighbor is not known, Address Resolution Protocol (ARP) is triggered during packet forwarding.<br>• The **per-vrf** keyword configures the same label to be used for all the routes advertised from a unique VRF. |
| Step 6 | **address-family** { **ipv4** \| **ipv6** } **unicast**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-vrf)# address-family ipv4 unicast` | Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.<br>To see a list of all the possible keywords and arguments for this command, use the CLI help (?). |
| Step 7 | **network** { *ip-address* / *prefix-length* \| *ip-address mask* }<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-af)#` | Originates a network prefix in the address family table in the VRF context. |

| | Command or Action | Purpose |
|---|---|---|
| | `network 172.16.5.5` | |
| Step 8 | **aggregate-address** *address* / *mask-length*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-af)# aggregate-address 10.0.0.0/24` | Configures aggregation in the VRF address family context to summarize routing information to reduce the state maintained in the core. This summarization introduces some inefficiency in the PE edge, because an additional lookup is required to determine the ultimate next hop for a packet. When configured, a summary prefix is advertised instead of a set of component prefixes, which are more specifics of the aggregate. The PE advertises only one label for the aggregate. Because component prefixes could have different next hops to CEs, an additional lookup has to be performed during data forwarding. |
| Step 9 | **exit**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-af)# exit` | Exits the current configuration mode. |
| Step 10 | **neighbor** *ip-address*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf)# neighbor 10.0.0.0` | Configures a CE neighbor. The *ip-address* argument must be a private address. |
| Step 11 | **remote-as** *as-number*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr)# remote-as 2` | Configures the remote AS for the CE neighbor. |
| Step 12 | **password** { **clear** \| **encrypted** } *password*<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr)# password encrypted 234xyz` | Enable Message Digest 5 (MD5) authentication on a TCP connection between two BGP neighbors. |
| Step 13 | **ebgp-multihop** [ *ttl-value* ]<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr)# ebgp-multihop 55` | Configures the CE neighbor to accept and attempt BGP connections to external peers residing on networks that are not directly connected. |

| | Command or Action | Purpose |
|---|---|---|
| **Step 14** | Do one of the following:<br>• **address-family** { **ipv4** \| **ipv6** } **unicast**<br>• **address-family {ipv4 {unicast \| labeled-unicast} \| ipv6 unicast}**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-vrf)# address-family ipv4 unicast` | Specifies either an IPv4 or IPv6 address family unicast and enters address family configuration submode.<br>To see a list of all the possible keywords and arguments for this command, use the CLI help (?). |
| **Step 15** | **site-of-origin** [ *as-number* **:** *nn* \| *ip-address* **:** *nn* ]<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr-af)# site-of-origin 234:111` | Configures the site-of-origin (SoO) extended community. Routes that are learned from this CE neighbor are tagged with the SoO extended community before being advertised to the rest of the PEs. SoO is frequently used to detect loops when as-override is configured on the PE router. If the prefix is looped back to the same site, the PE detects this and does not send the update to the CE. |
| **Step 16** | **as-override**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr-af)# as-override` | Configures AS override on the PE router. This causes the PE router to replace the CE's ASN with its own (PE) ASN.<br>**Note** This loss of information could lead to routing loops; to avoid loops caused by as-override, use it in conjunction with site-of-origin. |
| **Step 17** | **allowas-in** [ *as-occurrence-number* ]<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr-af)# allowas-in 5` | Allows an AS path with the PE autonomous system number (ASN) a specified number of times.<br>Hub and spoke VPN networks need the looping back of routing information to the HUB PE through the HUB CE. When this happens, due to the presence of the PE ASN, the looped-back information is dropped by the HUB PE. To avoid this, use the **allowas-in** command to allow prefixes even if they have the PEs ASN up to the specified number of times. |
| **Step 18** | **route-policy** *route-policy-name* **in**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr-af)# route-policy pe_ce_in_policy in` | Specifies a routing policy for an inbound route. The policy can be used to filter routes or modify route attributes. |

| | Command or Action | Purpose |
|---|---|---|
| **Step 19** | **route-policy** *route-policy-name* **out**<br>**Example:**<br>`RP/0/RP0/CPU0:router(config-bgp-vrf-nbr-af)# route-policy pe_ce_out_policy out` | Specifies a routing policy for an outbound route. The policy can be used to filter routes or modify route attributes. |
| **Step 20** | Use the **commit** or **end** command. | **commit** —Saves the configuration changes and remains within the configuration session.<br>**end** —Prompts user to take one of these actions:<br>• **Yes** — Saves configuration changes and exits the configuration session.<br>• **No** —Exits the configuration session without committing the configuration changes.<br>• **Cancel** —Remains in the configuration session, without committing the configuration changes. |

# Resetting an eBGP Session Immediately Upon Link Failure

By default, if a link goes down, all BGP sessions of any directly adjacent external peers are immediately reset. Use the **bgp fast-external-fallover disable** command to disable automatic resetting. Turn the automatic reset back on using the **no bgp fast-external-fallover disable** command.

eBGP sessions flap when the node reaches 3500 eBGP sessions with BGP timer values set as 10 and 30. To support more than 3500 eBGP sessions, increase the packet rate by using the **lpts pifib hardware police location** *location-id* command. Following is a sample configuration to increase the eBGP sessions:

```
RP/0/RP0/CPU0:router#configure
RP/0/RP0/CPU0:router(config)#lpts pifib hardware police location 0/2/CPU0
RP/0/RP0/CPU0:router(config-pifib-policer-per-node)#flow bgp configured rate 4000
RP/0/RP0/CPU0:router(config-pifib-policer-per-node)#flow bgp known rate 4000
RP/0/RP0/CPU0:router(config-pifib-policer-per-node)#flow bgp default rate 4000
RP/0/RP0/CPU0:router(config-pifib-policer-per-node)#commit
```

# Information about Implementing BGP

To implement BGP, you need to understand the following concepts:

## BGP Router Identifier

For BGP sessions between neighbors to be established, BGP must be assigned a router ID. The router ID is sent to BGP peers in the OPEN message when a BGP session is established.

BGP attempts to obtain a router ID in the following ways (in order of preference):

- By means of the address configured using the **bgp router-id** command in router configuration mode.

- By using the highest IPv4 address on a loopback interface in the system if the router is booted with saved loopback address configuration.
- By using the primary IPv4 address of the first loopback address that gets configured if there are not any in the saved configuration.

If none of these methods for obtaining a router ID succeeds, BGP does not have a router ID and cannot establish any peering sessions with BGP neighbors. In such an instance, an error message is entered in the system log, and the **show bgp summary** command displays a router ID of 0.0.0.0. After BGP has obtained a router ID, it continues to use it even if a better router ID becomes available. This usage avoids unnecessary flapping for all BGP sessions. However, if the router ID currently in use becomes invalid (because the interface goes down or its configuration is changed), BGP selects a new router ID (using the rules described) and all established peering sessions are reset.

**Note** We strongly recommend that the **bgp router-id** command is configured to prevent unnecessary changes to the router ID (and consequent flapping of BGP sessions).

## BGP Default Limits

BGP imposes maximum limits on the number of neighbors that can be configured on the router and on the maximum number of prefixes that are accepted from a peer for a given address family. This limitation safeguards the router from resource depletion caused by misconfiguration, either locally or on the remote neighbor. The following limits apply to BGP configurations:

- The default maximum number of peers that can be configured is 4000. The default can be changed using the **bgp maximum neighbor** command. The *limit* range is 1–15000. Any attempt to configure additional peers beyond the maximum limit or set the maximum limit to a number that is less than the number of peers that are currently configured will fail.
- To prevent a peer from flooding BGP with advertisements, a limit is placed on the number of prefixes that are accepted from a peer for each supported address family. The default limits can be overridden through configuration of the maximum-prefix *limit* command for the peer for the appropriate address family. The following default limits are used if the user does not configure the maximum number of prefixes for the address family:
    - 512K (524,288) prefixes for IPv4 unicast
    - 128K (131,072) prefixes for IPv6 unicast
    - 512K (524,288) prefixes for VPNv4 unicast

  A cease notification message is sent to the neighbor and the peering with the neighbor is terminated when the number of prefixes that are received from the peer for a given address family exceeds the maximum limit (either set by default or configured by the user) for that address family.

  It is possible that the maximum number of prefixes for a neighbor for a given address family has been configured after the peering with the neighbor has been established and some prefixes have already been received from the neighbor for that address family. A cease notification message is sent to the neighbor and peering with the neighbor is terminated immediately after the configuration if the configured maximum number of prefixes is fewer than the number of prefixes that have already been received from the neighbor for the address family.

# BGP Attributes and Operators

This table summarizes the BGP attributes and operators per attach points.

***Table 4: BGP Attributes and Operators***

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| aggregation | as-path | in<br>is-local<br>length<br>neighbor-is<br>originates-from<br>passes-through<br>unique-length | — |
| | as-path-length | is, ge, le, eq | — |
| | as-path-unique-length | is, ge, le, eq | — |
| | community | is-empty<br>matches-any<br>matches-every | set<br>set additive<br>delete in<br>delete not in<br>delete all |
| | destination | in | — |
| | extcommunity cost | — | set<br>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 |

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| allocate-label | as-path | in<br>is-local<br>length<br>neighbor-is<br>originates-from<br>passes-through<br>unique-length | — |
| | as-path-length | is, ge, le, eq | — |
| | as-path-unique-length | is, ge, le, eq | — |
| | community | is-empty<br>matches-any<br>matches-every | — |
| | destination | in | — |
| | label | — | set |
| | local-preference | is, ge, le, eq | — |
| | med | is, eg, ge, le | — |
| | next-hop | in | — |
| | origin | is | — |
| | source | in | — |
| clear-policy | as-path | in<br>is-local<br>length<br>neighbor-is<br>originates-from<br>passes-through<br>unique-length | — |
| | as-path-length | is, ge, le, eq | — |
| | as-path-unique-length | is, ge, le, eq | — |

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| dampening | as-path | in<br>is-local<br>length<br>neighbor-is<br>originates-from<br>passes-through<br>unique-length | — |
| | as-path-length | is, ge, le, eq | — |
| | as-path-unique-length | is, ge, le, eq | — |
| | community | is-empty<br>matches-any<br>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 | — |
| debug | destination | in | — |
| default originate | med | — | set<br>set +<br>set - |
| | rib-has-route | in | — |

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| neighbor-in | as-path | in<br>is-local<br>length<br>NA<br>neighbor-is<br>originates-from<br>passes-through<br>unique-length | prepend<br>prepend most-recent<br>remove as-path private-as<br>replace |
| | as-path-length | is, ge, le, eq | — |
| | as-path-unique-length | is, ge, le, eq | — |
| | communitycommunity with 'peeras' | is-empty<br>matches-any<br>matches-every | set<br>set additive<br>delete-in<br>delete-not-in<br>delete-all |
| | destination | in | — |
| | extcommunity cost | — | set<br>set additive |
| | extcommunity rt | is-empty<br>matches-any<br>matches-every<br>matches-within | set<br>additive<br>delete-in<br>delete-not-in<br>delete-all |
| | extcommunity soo | is-empty<br>matches-any<br>matches-every<br>matches-within | — |
| | local-preference | is, ge, le, eq | set |
| | med | is, eg, ge, le | set<br>set +<br>set - |

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| | next-hop | in | set<br>set peer address |
| | origin | is | set |
| | route-aggregated | route-aggregated | NA |
| | source | in | — |
| | weight | — | set |

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

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| | | | set<br>set +<br>set -<br>set max-unreachable<br>set igp-cost |
| | next-hop | in | set<br>set self |
| | origin | is | set |
| | path-type | is | — |
| | rd | in | — |
| | route-aggregated | route-aggregated | — |
| | source | in | — |
| | unsuppress-route | — | unsuppress-route |
| | vpn-distinguisher | — | set |
| neighbor-orf | orf-prefix | in | n/a |

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

| Attach Point | Attribute | Match | Set |
|---|---|---|---|
| redistribute | as-path | — | prepend |
| | community | — | set<br>set additive<br>delete in<br>delete not in<br>delete all |
| | destination | in | — |
| | extcommunity cost | — | setset additive |
| | local-preference | — | set |
| | med | — | set<br>set+<br>set- |
| | next-hop | in | set |
| | origin | — | set |
| | mpls-label | route-has-label | — |
| | route-type | is | — |
| | tag | is, eq, ge, le | — |
| | weight | — | set |
| retain-rt | extcommunity rt | is-empty<br>matches-any<br>matches-every<br>matches-within | — |

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

Some BGP route attributes are inaccessible from some BGP attach points for various reasons. For example, the **set med igp-cost only** command makes sense when there is a configured igp-cost to provide a source value.

This table summarizes which operations are valid and where they are valid.

*Table 5: Restricted BGP Operations by Attach Point*

| Command | import | export | aggregation | redistribution |
|---|---|---|---|---|
| prepend as-path most-recent | eBGP only | eBGP only | n/a | n/a |
| replace as-path | eBGP only | eBGP only | n/a | n/a |
| set med igp-cost | forbidden | eBGP only | forbidden | forbidden |
| set weight | n/a | forbidden | n/a | n/a |
| suppress | forbidden | forbidden | n/a | forbidden |

## BGP Best Path Algorithm

BGP routers typically receive multiple paths to the same destination. The BGP best-path algorithm determines the best path to install in the IP routing table and to use for forwarding traffic. This section describes the Cisco IOS XR software implementation of BGP best-path algorithm, as specified in Section 9.1 of the Internet Engineering Task Force (IETF) Network Working Group draft-ietf-idr-bgp4-24.txt document.

The BGP best-path algorithm implementation is in three parts:

- Part 1—Compares two paths to determine which is better.
- Part 2—Iterates over all paths and determines which order to compare the paths to select the overall best path.
- Part 3—Determines whether the old and new best paths differ enough so that the new best path should be used.

**Note** The order of comparison determined by Part 2 is important because the comparison operation is not transitive; that is, if three paths, A, B, and C exist, such that when A and B are compared, A is better, and when B and C are compared, B is better, it is not necessarily the case that when A and C are compared, A is better. This nontransitivity arises because the multi exit discriminator (MED) is compared only among paths from the same neighboring autonomous system (AS) and not among all paths.

### Comparing Pairs of Paths

Perform the following steps to compare two paths and determine the better path:

1. If either path is invalid (for example, a path has the maximum possible MED value or it has an unreachable next hop), then the other path is chosen (provided that the path is valid).
2. If the paths have unequal pre-bestpath cost communities, the path with the lower pre-bestpath cost community is selected as the best path.
3. If the paths have unequal weights, the path with the highest weight is chosen.

> **Note** The weight is entirely local to the router, and can be set with the **weight** command or using a routing policy.

4. If the paths have unequal local preferences, the path with the higher local preference is chosen.

> **Note** If a local preference attribute was received with the path or was set by a routing policy, then that value is used in this comparison. Otherwise, the default local preference value of 100 is used. The default value can be changed using the **bgp default local-preference** command.

5. If one of the paths is a redistributed path, which results from a **redistribute** or **network** command, then it is chosen. Otherwise, if one of the paths is a locally generated aggregate, which results from an **aggregate-address** command, it is chosen.

> **Note** Step 1 through Step 4 implement the "Path Selection with BGP"of RFC 1268.

6. If the paths have unequal AS path lengths, the path with the shorter AS path is chosen. This step is skipped if **bgp bestpath as-path ignore** command is configured.

> **Note** When calculating the length of the AS path, confederation segments are ignored, and AS sets count as 1.

> **Note** eiBGP specifies internal and external BGP multipath peers. eiBGP allows simultaneous use of internal and external paths.

7. If the paths have different origins, the path with the lower origin is selected. Interior Gateway Protocol (IGP) is considered lower than EGP, which is considered lower than INCOMPLETE.

8. If appropriate, the MED of the paths is compared. If they are unequal, the path with the lower MED is chosen.

   A number of configuration options exist that affect whether or not this step is performed. In general, the MED is compared if both paths were received from neighbors in the same AS; otherwise the MED comparison is skipped. However, this behavior is modified by certain configuration options, and there are also some corner cases to consider.

   If the **bgp bestpath med always** command is configured, then the MED comparison is always performed, regardless of neighbor AS in the paths. Otherwise, MED comparison depends on the AS paths of the two paths being compared, as follows:

   - If a path has no AS path or the AS path starts with an AS_SET, then the path is considered to be internal, and the MED is compared with other internal paths.
   - If the AS path starts with an AS_SEQUENCE, then the neighbor AS is the first AS number in the sequence, and the MED is compared with other paths that have the same neighbor AS.

- If the AS path contains only confederation segments or starts with confederation segments followed by an AS_SET, then the MED is not compared with any other path unless the **bgp bestpath med confed** command is configured. In that case, the path is considered internal and the MED is compared with other internal paths.
- If the AS path starts with confederation segments followed by an AS_SEQUENCE, then the neighbor AS is the first AS number in the AS_SEQUENCE, and the MED is compared with other paths that have the same neighbor AS.

**Note** If no MED attribute was received with the path, then the MED is considered to be 0 unless the **bgp bestpath med missing-as-worst** command is configured. In that case, if no MED attribute was received, the MED is considered to be the highest possible value.

9. If one path is received from an external peer and the other is received from an internal (or confederation) peer, the path from the external peer is chosen.
10. If the paths have different IGP metrics to their next hops, the path with the lower IGP metric is chosen.
11. If the paths have unequal IP cost communities, the path with the lower IP cost community is selected as the best path.
12. If all path parameters in Step 1 through Step 10 are the same, then the router IDs are compared. If the path was received with an originator attribute, then that is used as the router ID to compare; otherwise, the router ID of the neighbor from which the path was received is used. If the paths have different router IDs, the path with the lower router ID is chosen.

**Note** Where the originator is used as the router ID, it is possible to have two paths with the same router ID. It is also possible to have two BGP sessions with the same peer router, and therefore receive two paths with the same router ID.

13. If the paths have different cluster lengths, the path with the shorter cluster length is selected. If a path was not received with a cluster list attribute, it is considered to have a cluster length of 0.
14. Finally, the path received from the neighbor with the lower IP address is chosen. Locally generated paths (for example, redistributed paths) are considered to have a neighbor IP address of 0.

## Order of Comparisons

The second part of the BGP best-path algorithm implementation determines the order in which the paths should be compared. The order of comparison is determined as follows:

1. The paths are partitioned into groups such that within each group the MED can be compared among all paths. The same rules as in #unique_126 are used to determine whether MED can be compared between any two paths. Normally, this comparison results in one group for each neighbor AS. If the **bgp bestpath med always** command is configured, then there is just one group containing all the paths.
2. The best path in each group is determined. Determining the best path is achieved by iterating through all paths in the group and keeping track of the best one seen so far. Each path is compared with the best-so-far, and if it is better, it becomes the new best-so-far and is compared with the next path in the group.

3. A set of paths is formed containing the best path selected from each group in Step 2. The overall best path is selected from this set of paths, by iterating through them as in Step 2.

### Best Path Change Suppression

The third part of the implementation is to determine whether the best-path change can be suppressed or not—whether the new best path should be used, or continue using the existing best path. The existing best path can continue to be used if the new one is identical to the point at which the best-path selection algorithm becomes arbitrary (if the router-id is the same). Continuing to use the existing best path can avoid churn in the network.

**Note** This suppression behavior does not comply with the IETF Networking Working Group draft-ietf-idr-bgp4-24.txt document, but is specified in the IETF Networking Working Group draft-ietf-idr-avoid-transition-00.txt document.

The suppression behavior can be turned off by configuring the **bgp bestpath compare-routerid** command. If this command is configured, the new best path is always preferred to the existing one.

Otherwise, the following steps are used to determine whether the best-path change can be suppressed:

1. If the existing best path is no longer valid, the change cannot be suppressed.
2. If either the existing or new best paths were received from internal (or confederation) peers or were locally generated (for example, by redistribution), then the change cannot be suppressed. That is, suppression is possible only if both paths were received from external peers.
3. If the paths were received from the same peer (the paths would have the same router-id), the change cannot be suppressed. The router ID is calculated using rules in #unique_126.
4. If the paths have different weights, local preferences, origins, or IGP metrics to their next hops, then the change cannot be suppressed. Note that all these values are calculated using the rules in #unique_126.
5. If the paths have different-length AS paths and the **bgp bestpath as-path ignore** command is not configured, then the change cannot be suppressed. Again, the AS path length is calculated using the rules in #unique_126.
6. If the MED of the paths can be compared and the MEDs are different, then the change cannot be suppressed. The decision as to whether the MEDs can be compared is exactly the same as the rules in #unique_126, as is the calculation of the MED value.
7. If all path parameters in Step 1 through Step 6 do not apply, the change can be suppressed.

## BGP Update Generation and Update Groups

The BGP Update Groups feature separates BGP update generation from neighbor configuration. The BGP Update Groups feature introduces an algorithm that dynamically calculates BGP update group membership based on outbound routing policies. This feature does not require any configuration by the network operator. Update group-based message generation occurs automatically and independently.

## BGP Update Group

When a change to the configuration occurs, the router automatically recalculates update group memberships and applies the changes.

For the best optimization of BGP update group generation, we recommend that the network operator keeps outbound routing policy the same for neighbors that have similar outbound policies. This feature contains commands for monitoring BGP update groups.

## BGP Cost Community Reference

The cost community attribute is applied to internal routes by configuring the **set extcommunity cost** command in a route policy. The cost community set clause is configured with a cost community ID number (0–255) and cost community number (0–4294967295). The cost community number determines the preference for the path. The path with the lowest cost community number is preferred. Paths that are not specifically configured with the cost community number are assigned a default cost community number of 2147483647 (the midpoint between 0 and 4294967295) and evaluated by the best-path selection process accordingly. When two paths have been configured with the same cost community number, the path selection process prefers the path with the lowest cost community ID. The cost-extended community attribute is propagated to iBGP peers when extended community exchange is enabled.

The following commands include the **route-policy** keyword, which you can use to apply a route policy that is configured with the cost community set clause:

- **aggregate-address**
- **redistribute**
- **network**

## BGP Next Hop Reference

Event notifications from the RIB are classified as critical and noncritical. Notifications for critical and noncritical events are sent in separate batches. BGP is notified when any of the following events occurs:

- Next hop becomes unreachable
- Next hop becomes reachable
- Fully recursed IGP metric to the next hop changes
- First hop IP address or first hop interface change
- Next hop becomes connected
- Next hop becomes unconnected
- Next hop becomes a local address
- Next hop becomes a nonlocal address

**Note** Reachability and recursed metric events trigger a best-path recalculation.

However, a noncritical event is sent along with the critical events if the noncritical event is pending and there is a request to read the critical events.

- Critical events are related to the reachability (reachable and unreachable), connectivity (connected and unconnected), and locality (local and nonlocal) of the next hops. Notifications for these events are not delayed.

- Noncritical events include only the IGP metric changes. These events are sent at an interval of 3 seconds. A metric change event is batched and sent 3 seconds after the last one was sent.

BGP is notified when any of the following events occurs:

- Next hop becomes unreachable
- Next hop becomes reachable
- Fully recursed IGP metric to the next hop changes
- First hop IP address or first hop interface change
- Next hop becomes connected
- Next hop becomes unconnected
- Next hop becomes a local address
- Next hop becomes a nonlocal address

**Note** Reachability and recursed metric events trigger a best-path recalculation.

The next-hop trigger delay for critical and noncritical events can be configured to specify a minimum batching interval for critical and noncritical events using the **nexthop trigger-delay** command. The trigger delay is address family dependent.

The BGP next-hop tracking feature allows you to specify that BGP routes are resolved using only next hops whose routes have the following characteristics:

- To avoid the aggregate routes, the prefix length must be greater than a specified value.
- The source protocol must be from a selected list, ensuring that BGP routes are not used to resolve next hops that could lead to oscillation.

This route policy filtering is possible because RIB identifies the source protocol of route that resolved a next hop as well as the mask length associated with the route. The **nexthop route-policy** command is used to specify the route-policy.

**Next Hop as the IPv6 Address of Peering Interface**

BGP can carry IPv6 prefixes over an IPv4 session. The next hop for the IPv6 prefixes can be set through a nexthop policy. In the event that the policy is not configured, the nexthops are set as the IPv6 address of the peering interface (IPv6 neighbor interface or IPv6 update source interface, if any one of the interfaces is configured).

If the nexthop policy is not configured and neither the IPv6 neighbor interface nor the IPv6 update source interface is configured, the next hop is the IPv4 mapped IPv6 address.

**Scoped IPv4/VPNv4 Table Walk**

To determine which address family to process, a next-hop notification is received by first de-referencing the gateway context associated with the next hop, then looking into the gateway context to determine which address families are using the gateway context. The IPv4 unicast and VPNv4 unicast address families share the same gateway context, because they are registered with the IPv4 unicast table in the RIB. As a result, both the global IPv4 unicast table and the VPNv4 table are is processed when an IPv4 unicast next-hop notification is received from the RIB. A mask is maintained in the next hop, indicating if whether the next hop belongs to IPv4 unicast or VPNv4 unicast, or both. This scoped table walk localizes the processing in the appropriate address family table.

**Reordered Address Family Processing**

The software walks address family tables based on the numeric value of the address family. When a next-hop notification batch is received, the order of address family processing is reordered to the following order:

- IPv4 tunnel
- VPNv4 unicast
- VPNv6 unicast
- IPv4 labeled unicast
- IPv4 unicast
- IPv4 MDT
- IPv6 unicast
- IPv6 labeled unicast
- IPv4 tunnel
- VPNv4 unicast
- IPv4 unicast
- IPv6 unicast

**New Thread for Next-Hop Processing**

The critical-event thread in the spkr process handles only next-hop, Bidirectional Forwarding Detection (BFD), and fast-external-failover (FEF) notifications. This critical-event thread ensures that BGP convergence is not adversely impacted by other events that may take a significant amount of time.

**show, clear, and debug Commands**

The **show bgp nexthops** command provides statistical information about next-hop notifications, the amount of time spent in processing those notifications, and details about each next hop registered with the RIB. The **clear bgp nexthop performance-statistics** command ensures that the cumulative statistics associated with the processing part of the next-hop **show** command can be cleared to help in monitoring. The **clear bgp nexthop registration** command performs an asynchronous registration of the next hop with the RIB.

The **debug bgp nexthop** command displays information on next-hop processing. The **out** keyword provides debug information only about BGP registration of next hops with RIB. The **in** keyword displays debug information about next-hop notifications received from RIB. The **out** keyword displays debug information about next-hop notifications sent to the RIB.

## BGP Nonstop Routing Reference

BGP NSR provides nonstop routing during the following events:

- Route processor switchover
- Process crash or process failure of BGP or TCP

**Note** BGP NSR is enabled by default. Use the **nsr disable** command to turn off BGP NSR. The **no nsr disable** command can also be used to turn BGP NSR back on if it has been disabled.

In case of process crash or process failure, NSR will be maintained only if **nsr process-failures switchover** command is configured. In the event of process failures of active instances, the **nsr process-failures switchover** configures failover as a recovery action and switches over to a standby route processor (RP) or a standby distributed route processor (DRP) thereby maintaining NSR. An example of the configuration command is RP/0/RSP0/CPU0:router(config) # nsr process-failures switchover

The **nsr process-failures switchover** command maintains both the NSR and BGP sessions in the event of a BGP or TCP process crash. Without this configuration, BGP neighbor sessions flap in case of a BGP or TCP process crash. This configuration does not help if the BGP or TCP process is restarted in which case the BGP neighbors are expected to flap.

When the *l2vpn_mgr* process is restarted, the NSR client (te-control) flaps between the **Ready** and **Not Ready** state. This is the expected behavior and there is no traffic loss.

During route processor switchover and In-Service System Upgrade (ISSU), NSR is achieved by stateful switchover (SSO) of both TCP and BGP.

NSR does not force any software upgrades on other routers in the network, and peer routers are not required to support NSR.

When a route processor switchover occurs due to a fault, the TCP connections and the BGP sessions are migrated transparently to the standby route processor, and the standby route processor becomes active. The existing protocol state is maintained on the standby route processor when it becomes active, and the protocol state does not need to be refreshed by peers.

Events such as soft reconfiguration and policy modifications can trigger the BGP internal state to change. To ensure state consistency between active and standby BGP processes during such events, the concept of post-it is introduced that act as synchronization points.

BGP NSR provides the following features:

- NSR-related alarms and notifications
- Configured and operational NSR states are tracked separately
- NSR statistics collection
- NSR statistics display using **show** commands
- XML schema support
- Auditing mechanisms to verify state synchronization between active and standby instances
- CLI commands to enable and disable NSR

# BGP Route Reflectors Reference

#unique_52 unique_52_Connect_42_fig_ED746D3CC16E445F85EB3B1CE06B3767 illustrates a simple iBGP configuration with three iBGP speakers (routers A, B, and C). Without route reflectors, when Router A receives a route from an external neighbor, it must advertise it to both routers B and C. Routers B and C do not readvertise the iBGP learned route to other iBGP speakers because the routers do not pass on routes learned from internal neighbors to other internal neighbors, thus preventing a routing information loop.

With route reflectors, all iBGP speakers need not be fully meshed because there is a method to pass learned routes to neighbors. In this model, an iBGP peer is configured to be a route reflector responsible for passing iBGP learned routes to a set of iBGP neighbors. In #unique_52 unique_52_Connect_42_fig_3980C23832D84D43BB58222F040E5A96 , Router B is configured as a route reflector. When the route reflector receives routes advertised from Router A, it advertises them to Router C, and vice versa. This scheme eliminates the need for the iBGP session between routers A and C.

The internal peers of the route reflector are divided into two groups: client peers and all other routers in the autonomous system (nonclient peers). A route reflector reflects routes between these two groups. The route reflector and its client peers form a *cluster*. The nonclient peers must be fully meshed with each other, but the client peers need not be fully meshed. The clients in the cluster do not communicate with iBGP speakers outside their cluster.

#unique_52 unique_52_Connect_42_fig_9026266F2853489B94D037A781853752 illustrates a more complex route reflector scheme. Router A is the route reflector in a cluster with routers B, C, and D. Routers E, F, and G are fully meshed, nonclient routers.

When the route reflector receives an advertised route, depending on the neighbor, it takes the following actions:

- A route from an external BGP speaker is advertised to all clients and nonclient peers.
- A route from a nonclient peer is advertised to all clients.
- A route from a client is advertised to all clients and nonclient peers. Hence, the clients need not be fully meshed.

Along with route reflector-aware BGP speakers, it is possible to have BGP speakers that do not understand the concept of route reflectors. They can be members of either client or nonclient groups, allowing an easy and gradual migration from the old BGP model to the route reflector model. Initially, you could create a single cluster with a route reflector and a few clients. All other iBGP speakers could be nonclient peers to the route reflector and then more clusters could be created gradually.

An autonomous system can have multiple route reflectors. A route reflector treats other route reflectors just like other iBGP speakers. A route reflector can be configured to have other route reflectors in a client group or nonclient group. In a simple configuration, the backbone could be divided into many clusters. Each route reflector would be configured with other route reflectors as nonclient peers (thus, all route reflectors are fully meshed). The clients are configured to maintain iBGP sessions with only the route reflector in their cluster.

Usually, a cluster of clients has a single route reflector. In that case, the cluster is identified by the router ID of the route reflector. To increase redundancy and avoid a single point of failure, a cluster might have more than one route reflector. In this case, all route reflectors in the cluster must be configured with the cluster ID so that a route reflector can recognize updates from route reflectors in the same cluster. All route reflectors serving a cluster should be fully meshed and all of them should have identical sets of client and nonclient peers.

By default, the clients of a route reflector are not required to be fully meshed and the routes from a client are reflected to other clients. However, if the clients are fully meshed, the route reflector need not reflect routes to clients.

As the iBGP learned routes are reflected, routing information may loop. The route reflector model has the following mechanisms to avoid routing loops:

- Originator ID is an optional, nontransitive BGP attribute. It is a 4-byte attributed created by a route reflector. The attribute carries the router ID of the originator of the route in the local autonomous system. Therefore, if a misconfiguration causes routing information to come back to the originator, the information is ignored.
- Cluster-list is an optional, nontransitive BGP attribute. It is a sequence of cluster IDs that the route has passed. When a route reflector reflects a route from its clients to nonclient peers, and vice versa, it appends the local cluster ID to the cluster-list. If the cluster-list is empty, a new cluster-list is created. Using this attribute, a route reflector can identify if routing information is looped back to the same cluster due to misconfiguration. If the local cluster ID is found in the cluster-list, the advertisement is ignored.

## iBGP Multipath Load Sharing Reference

When there are multiple border BGP routers having reachability information heard over eBGP, if no local policy is applied, the border routers will choose their eBGP paths as best. They advertise that bestpath inside the ISP network. For a core router, there can be multiple paths to the same destination, but it will select only one path as best and use that path for forwarding. iBGP multipath load sharing adds the ability to enable load sharing among multiple equi-distant paths. Configuring multiple iBGP best paths enables a router to evenly share the traffic destined for a particular site. The iBGP Multipath Load Sharing feature functions similarly in a Multiprotocol Label Switching (MPLS) Virtual Private Network (VPN) with a service provider backbone.

For multiple paths to the same destination to be considered as multipaths, the following criteria must be met:

- All attributes must be the same. The attributes include weight, local preference, autonomous system path (entire attribute and not just length), origin code, Multi Exit Discriminator (MED), and Interior Gateway Protocol (iGP) distance.
- The next hop router for each multipath must be different.

Even if the criteria are met and multiple paths are considered multipaths, the BGP speaking router designates one of the multipaths as the best path and advertises this best path to its neighbors.

**Note**

- Overwriting of next-hop calculation for multipath prefixes is not allowed. The **next-hop-unchanged multipath** command disables overwriting of next-hop calculation for multipath prefixes.
- The ability to ignore as-path onwards while computing multipath is added. The **bgp multipath as-path ignore onwards** command ignores as-path onwards while computing multipath.

## L3VPN iBGP PE-CE Reference

When BGP is used as the provider edge (PE) or the customer edge (CE) routing protocol, the peering sessions are configured as external peering between the VPN provider autonomous system (AS) and the customer network autonomous system. The L3VPN iBGP PE-CE feature enables the PE and CE devices to exchange Border Gateway Protocol (BGP) routing information by peering as internal Border Gateway Protocol (iBGP) instead of the widely-used external BGP peering between the PE and the CE. This mechanism applies at each PE device where a VRF-based CE is configured as iBGP. This eliminates the need for service providers (SPs) to configure autonomous system override for the CE. With this feature enabled, there is no need to configure the virtual private network (VPN) sites using different autonomous systems.

The **neighbor internal-vpn-client** command enables PE devices to make an entire VPN cloud act as an internal VPN client to the CE devices. These CE devices are connected internally to the VPN cloud through the iBGP PE-CE connection inside the VRF. After this connection is established, the PE device encapsulates the CE-learned path into an attribute called ATTR_SET and carries it in the iBGP-sourced path throughout the VPN core to the remote PE device. At the remote PE device, this attribute is assigned with individual attributes and the source CE path is extracted and sent to the remote CE devices.

ATTR_SET is an optional transitive attribute that carries the CE path attributes received. The ATTR_SET attribute is encoded inside the BGP update message as follows:

```
+------------------------------+
| Attr Flags (O|T) Code = 128  |
+------------------------------+
| Attr. Length (1 or 2 octets) |
+------------------------------+
| Origin AS (4 octets)         |
+------------------------------+
| Path attributes (variable)   |
+------------------------------+
```

Origin AS is the AS of the VPN customer for which the ATTR_SET is generated. The minimum length of ATTR_SET is four bytes and the maximum is the maximum supported for a path attribute after taking into consideration the mandatory fields and attributes in the BGP update message. It is recommended that the maximum length is limited to 3500 bytes. ATTR_SET must not contain the following attributes: MP_REACH, MP_UNREACH, NEW_AS_PATH, NEW_AGGR, NEXT_HOP and ATTR_SET itself (ATTR_SET inside ATTR_SET). If these attributes are found inside the ATTR_SET, the ATTR_SET is considered invalid and the corresponding error handling mechanism is invoked.

## MPLS VPN Carrier Supporting Carrier

Carrier supporting carrier (CSC) is a term used to describe a situation in which one service provider allows another service provider to use a segment of its backbone network. The service provider that provides the segment of the backbone network to the other provider is called the *backbone carrier*. The service provider that uses the segment of the backbone network is called the *customer carrier*.

A backbone carrier offers Border Gateway Protocol and Multiprotocol Label Switching (BGP/MPLS) VPN services. The customer carrier can be either:

- An Internet service provider (ISP) (By definition, an ISP does not provide VPN service.)
- A BGP/MPLS VPN service provider

You can configure a CSC network to enable BGP to transport routes and MPLS labels between the backbone carrier provider edge (PE) routers and the customer carrier customer edge (CE) routers using multiple paths. The benefits of using BGP to distribute IPv4 routes and MPLS label routes are:

- BGP takes the place of an Interior Gateway Protocol (IGP) and Label Distribution Protocol (LDP) in a VPN routing and forwarding (VRF) table. You can use BGP to distribute routes and MPLS labels. Using a single protocol instead of two simplifies the configuration and troubleshooting.
- BGP is the preferred routing protocol for connecting two ISPs, mainly because of its routing policies and ability to scale. ISPs commonly use BGP between two providers. This feature enables those ISPs to use BGP.

For detailed information on configuring MPLS VPN CSC with BGP, see the *Implementing MPLS Layer 3 VPNs on* module of the *MPLS Configuration Guide*.

## Per VRF and Per CE Label for IPv6 Provider Edge

The per VRF and per CE label for IPv6 feature makes it possible to save label space by allocating labels per default VRF or per CE nexthop.

All IPv6 Provider Edge (6PE) labels are allocated per prefix by default. Each prefix that belongs to a VRF instance is advertised with a single label, causing an additional lookup to be performed in the VRF forwarding table to determine the customer edge (CE) next hop for the packet.

However, use the **label-allocation-mode** command with the **per-ce** keyword or the **per-vrf** keyword to avoid the additional lookup on the PE router and conserve label space.

Use **per-ce** keyword to specify that the same label be used for all the routes advertised from a unique customer edge (CE) peer router. Use the **per-vrf** keyword to specify that the same label be used for all the routes advertised from a unique VRF.

## IPv6 Unicast Routing

Cisco provides complete Internet Protocol Version 6 (IPv6) unicast capability.

An IPv6 unicast address is an identifier for a single interface, on a single node. A packet that is sent to a unicast address is delivered to the interface identified by that address. Cisco IOS XR software supports the following IPv6 unicast address types:

- Global aggregatable address
- Site-local address
- Link-local address
- IPv4-compatible IPv6 address

For more information on IPv6 unicast addressing, refer the *IP Addresses and Services Configuration Guide*.

## Remove and Replace Private AS Numbers from AS Path in BGP

Private autonomous system numbers (ASNs) are used by Internet Service Providers (ISPs) and customer networks to conserve globally unique AS numbers. Private AS numbers cannot be used to access the global Internet because they are not unique. AS numbers appear in eBGP AS paths in routing updates. Removing private ASNs from the AS path is necessary if you have been using private ASNs and you want to access the global Internet.

Public AS numbers are assigned by InterNIC and are globally unique. They range from 1 to 64511. Private AS numbers are used to conserve globally unique AS numbers, and they range from 64512 to 65535. Private AS numbers cannot be leaked to a global BGP routing table because they are not unique, and BGP best path calculations require unique AS numbers. Therefore, it might be necessary to remove private AS numbers from an AS path before the routes are propagated to a BGP peer.

External BGP (eBGP) requires that globally unique AS numbers be used when routing to the global Internet. Using private AS numbers (which are not unique) would prevent access to the global Internet. The remove and replace private AS Numbers from AS Path in BGP feature allows routers that belong to a private AS to access the global Internet. A network administrator configures the routers to remove private AS numbers from the AS path contained in outgoing update messages and optionally, to replace those numbers with the ASN of the local router, so that the AS Path length remains unchanged.

The ability to remove and replace private AS numbers from the AS Path is implemented in the following ways:

- The **remove-private-as** command removes private AS numbers from the AS path even if the path contains both public and private ASNs.
- The **remove-private-as** command removes private AS numbers even if the AS path contains only private AS numbers. There is no likelihood of a 0-length AS path because this command can be applied to eBGP peers only, in which case the AS number of the local router is appended to the AS path.
- The **remove-private-as** command removes private AS numbers even if the private ASNs appear before the confederation segments in the AS path.
- The **replace-as** command replaces the private AS numbers being removed from the path with the local AS number, thereby retaining the same AS path length.

The feature can be applied to neighbors per address family (address family configuration mode). Therefore, you can apply the feature for a neighbor in one address family and not on another, affecting update messages on the outbound side for only the address family for which the feature is configured.

Use **show bgp neighbors** and **show bgp update-group** commands to verify that the that private AS numbers were removed or replaced.

## BGP Update Message Error Handling

The BGP UPDATE message error handling changes BGP behavior in handling error UPDATE messages to avoid session reset. Based on the approach described in IETF IDR *I-D:draft-ietf-idr-error-handling*, the Cisco IOS XR BGP UPDATE Message Error handling implementation classifies BGP update errors into various categories based on factors such as, severity, likelihood of occurrence of UPDATE errors, or type of attributes. Errors encountered in each category are handled according to the draft. Session reset will be avoided as much as possible during the error handling process. Error handling for some of the categories are controlled by configuration commands to enable or disable the default behavior.

According to the base BGP specification, a BGP speaker that receives an UPDATE message containing a malformed attribute is required to reset the session over which the offending attribute was received. This behavior is undesirable as a session reset would impact not only routes with the offending attribute, but also other valid routes exchanged over the session.

## BGP Error Handling and Attribute Filtering Syslog Messages

When a router receives a malformed update packet, an ios_msg of type ROUTING-BGP-3-MALFORM_UPDATE is printed on the console. This is rate limited to 1 message per minute across all neighbors. For malformed packets that result in actions "Discard Attribute" (A5) or "Local Repair" (A6), the ios_msg is printed only once per neighbor per action. This is irrespective of the number of malformed updates received since the neighbor last reached an "Established" state.

This is a sample BGP error handling syslog message:

```
%ROUTING-BGP-3-MALFORM_UPDATE : Malformed UPDATE message received from neighbor 13.0.3.50
- message length 90 bytes,
 error flags 0x00000840, action taken "TreatAsWithdraw".
Error details: "Error 0x00000800, Field "Attr-missing", Attribute 1 (Flags 0x00, Length 0),
 Data []"
```

This is a sample BGP attribute filtering syslog message for the "discard attribute" action:

```
[4843.46]RP/0/RP0/CPU0:Aug 21 17:06:17.919 : bgp[1037]: %ROUTING-BGP-5-UPDATE_FILTERED :
One or more attributes were filtered from UPDATE message received from neighbor 40.0.101.1
```

```
 - message length 173 bytes,
 action taken "DiscardAttr".
 Filtering details: "Attribute 16 (Flags 0xc0): Action "DiscardAttr"". NLRIs: [IPv4 Unicast]
 88.2.0.0/17
```

This is a sample BGP attribute filtering syslog message for the "treat-as-withdraw" action:

```
[391.01]RP/0/RP0/CPU0:Aug 20 19:41:29.243 : bgp[1037]: %ROUTING-BGP-5-UPDATE_FILTERED :
 One or more attributes were filtered from UPDATE message received from neighbor 40.0.101.1
 - message length 166 bytes,
 action taken "TreatAsWdr".
 Filtering details: "Attribute 4 (Flags 0xc0): Action "TreatAsWdr"". NLRIs: [IPv4 Unicast]
 88.2.0.0/17
```

## BGP-RIB Feedback Mechanism for Update Generation

The Border Gateway Protocol-Routing Information Base (BGP-RIB) feedback mechanism for update generation feature avoids premature route advertisements and subsequent packet loss in a network. This mechanism ensures that routes are installed locally, before they are advertised to a neighbor.

BGP waits for feedback from RIB indicating that the routes that BGP installed in RIB are installed in forwarding information base (FIB) before BGP sends out updates to the neighbors. RIB uses the the BCDL feedback mechanism to determine which version of the routes have been consumed by FIB, and updates the BGP with that version. BGP will send out updates of only those routes that have versions up to the version that FIB has installed. This selective update ensures that BGP does not send out premature updates resulting in attracting traffic even before the data plane is programmed after router reload, LC OIR, or flap of a link where an alternate path is made available.

To configure BGP to wait for feedback from RIB indicating that the routes that BGP installed in RIB are installed in FIB, before BGP sends out updates to neighbors, use the **update wait-install** command in router address-family IPv4 or router address-family VPNv4 configuration mode. The **show bgp**, **show bgp neighbors**, and **show bgp process performance-statistics** commands display the information from update wait-install configuration.

## Use-defined Martian Check

The solution allows disabling the Martian check for these IP address prefixes:

- IPv4 address prefixes
  - 0.0.0.0/8
  - 127.0.0.0/8
  - 224.0.0.0/4
- IPv6 address prefixes
  - ::
  - ::0002 - ::ffff
  - ::ffff:a.b.c.d
  - fe80:xxxx
  - ffxx:xxxx

CHAPTER 2

# EVPN Virtual Private Wire Service (VPWS)

The EVPN-VPWS is a BGP control plane solution for point-to-point services. It implements the signaling and encapsulation techniques for establishing an EVPN instance between a pair of PEs. It has the ability to forward traffic from one network to another without MAC lookup. The use of EVPN for VPWS eliminates the need for signaling single-segment and multi-segment PWs for point-to-point Ethernet services.

EVPN-VPWS single homed technology works on IP and MPLS core; IP core to support BGP and MPLS core for switching packets between the endpoints.

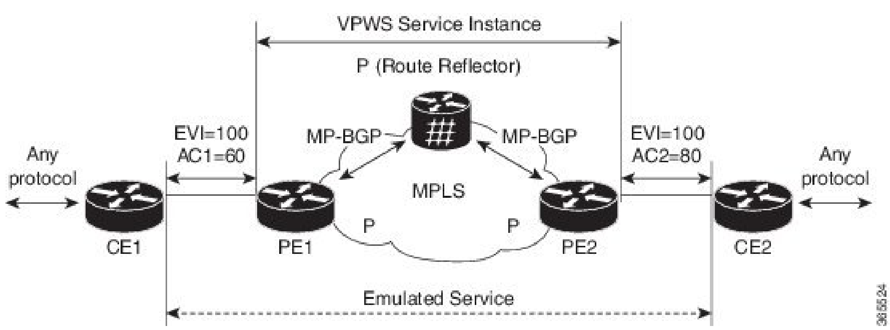
**Note** Other than enabling RTC (route target constraint) with `address-family ipv4 rtfilter` command, there is no separate configuration needed to enable RTC for BGP EVPN.



# Information About EVPN-VPWS Single Homed

The EVPN-VPWS single homed solution requires per EVI Ethernet Auto Discovery route. EVPN defines a new BGP Network Layer Reachability Information (NLRI) used to carry all EVPN routes. BGP Capabilities Advertisement used to ensure that two speakers support EVPN NLRI (AFI 25, SAFI 70) as per RFC 4760.

The architecture for EVPN VPWS is that the PEs run Multi-Protocol BGP in control-plane. The following image describes the EVPN-VPWS configuration:



- The VPWS service on PE1 requires the following three elements to be specified at configuration time:
  - The VPN ID (EVI)
  - The local AC identifier (AC1) that identifies the local end of the emulated service.
  - The remote AC identifier (AC2) that identifies the remote end of the emulated service.

  PE1 allocates a MPLS label per local AC for reachability.
- The VPWS service on PE2 is set in the same manner as PE1. The three same elements are required and the service configuration must be symmetric.

  PE2 allocates a MPLS label per local AC for reachability.
- PE1 advertise a single EVPN per EVI Ethernet AD route for each local endpoint (AC) to remote PEs with the associated MPLS label.

  PE2 performs the same task.
- On reception of EVPN per EVI EAD route from PE2, PE1 adds the entry to its local L2 RIB. PE1 knows the path list to reach AC2, for example, next hop is PE2 IP address and MPLS label for AC2.

  PE2 performs the same task.

# Configuring L2VPN EVPN Address Family Under BGP

Perform this task to configure L2VPN EVPN address family under BGP.

**Note** Other than enabling RTC (route target constraint) with `address-family ipv4 rtfilter` command, there is no separate configuration needed to enable RTC for BGP EVPN.

**Procedure**

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters the Global Configuration mode.

**Step 2** **router bgp** *autonomous-system-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Enters router configuration mode for the specified routing process.

**Step 3** **address-family l2vpn evpn**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family l2vpn evpn
RP/0/RP0/CPU0:router(config-bgp-af)# exit
```

Specifies the L2VPN address family and enters address family configuration mode.

**Step 4** **neighbor** *ip-address*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# neighbor 10.10.10.1
```

Adds the IP address of the neighbor in the specified autonomous system.

**Step 5** **address-family l2vpn evpn**

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-nbr)# address-family l2vpn evpn
```

Specifies the L2VPN address family of the neighbor and enters address family configuration mode.

**Step 6** Use the **commit** or **end** command.

**commit** - Saves the configuration changes and remains within the configuration session.

**end** - Prompts user to take one of these actions:

- **Yes** - Saves configuration changes and exits the configuration session.
- **No** - Exits the configuration session without committing the configuration changes.
- **Cancel** - Remains in the configuration mode, without committing the configuration changes.

# Configuring EVPN-VPWS

Perform this task to configure EVPN-VPWS.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters the Global Configuration mode.

**Step 2** **l2vpn**

**Example:**

```
RP/0/RP0/CPU0:router(config)# l2vpn
```

Enters Layer 2 VPN configuration mode.

**Step 3** **xconnect group** *group-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn)# xconnect group evpn-vpws
```

Configures a cross-connect group name using a free-format 32-character string.

**Step 4** **p2p** *xconnect-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc)# p2p evpn1
```

Enters P2P configuration submode.

**Step 5** **interface** *type interface-path-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-p2p)# interface TenGigE0/1/0/2
```

Specifies the interface type and instance.

**Step 6** **neighbor evpn evi** *vpn-id***target** *ac-id* **source** *ac-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-p2p)# neighbor evpn evi 100 target 12 source 10
```

Enables EVPN-VPWS endpoint on the p2p cross-connect.

**Step 7** Use the **commit** or **end** command.

**commit** - Saves the configuration changes and remains within the configuration session.

**end** - Prompts user to take one of these actions:

- **Yes** - Saves configuration changes and exits the configuration session.
- **No** - Exits the configuration session without committing the configuration changes.
- **Cancel** - Remains in the configuration mode, without committing the configuration changes.

# Configuring EVPN-VPWS: Example

The following example shows how to configure EVPN-VPWS service.

```
RP/0/RP0/CPU0:router# configure
RP/0/RP0/CPU0:router(config)# l2vpn
RP/0/RP0/CPU0:router(config-l2vpn)# xconnect group evpn-vpws
RP/0/RP0/CPU0:router(config-l2vpn-xc)# p2p evpn1
RP/0/RP0/CPU0:router(config-l2vpn-xc-p2p)# interface TenGigE0/1/0/12
RP/0/RP0/CPU0:router(config-l2vpn-xc-p2p)# neighbor evpn evi 100 target 12 source 10
```

CHAPTER 3

# BGP-based VPWS Autodiscovery

An important aspect of VPN technologies is the ability of network devices to automatically signal to other devices about an association with a particular VPN. Autodiscovery refers to the process of finding all the provider edge routers that participates in a given VPWS instance.

The two primary functions of the VPWS control plane are: auto-discovery and signaling. Both of these functions are accomplished with a single BGP Update advertisement.

When a VPWS cross-connect is configured with BGP auto-discovery and signaling enabled, BGP needs to distribute NLRI for the xconnect with the PE as the BGP next-hop and appropriate CE-ID. Additionally, the cross-connect is associated with one or more BGP export Route Targets (RTs) that are also distributed (along with NLRI).



## Configuring VPWS with BGP Autodiscovery and Signaling

Perform this task to configure BGP-based autodiscovery and signaling.

### Procedure

**Step 1** **configure**

**Example:**

```
RP/0/RP0/CPU0:router# configure
```

Enters the global configuration mode.

**Step 2** **l2vpn**

**Example:**

```
RP/0/RP0/CPU0:router(config)# l2vpn
```

Enters L2VPN configuration mode.

**Step 3** **xconnect group** *group name*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn)# xconnect group gr1
```

Enters configuration mode for the named xconnect group.

**Step 4** **mp2mp** *vpws-domain name*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc)# mp2mp mp1
```

Enters configuration mode for the named vpws domain.

**Step 5** **vpn-id** *vpn-id*

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-m2mp)# vpn-id 100
```

Specifies the identifier for the VPWS service.

**Step 6** **l2 encapsulation vlan**

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp)#l2-encapsulation vlan
```

Configure the L2 encapsulation for this L2VPN MP2MP Instance.

**Step 7** **autodiscovery bgp**

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp)#autodiscovery bgp
```

Enters BGP autodiscovery configuration mode where all BGP autodiscovery parameters are configured.

**Step 8** **rd** { *as-number:nn* | *ip-address:nn* | **auto** }

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp-ad)# rd auto
```

Specifies the route distinguisher (RD).

**Step 9** **route-target** { *as-number:nn* | *ip-address:nn* | **export** | **import** }

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp-ad)# route-target 500:99
```

Specifies the route target (RT).

**Step 10** **signaling-protocol bgp**

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp-ad)# signaling-protocol bgp
```

Enables BGP signaling, and enters the BGP signaling configuration submode where BGP signaling parameters are configured.

**Step 11** **ce-id** { *number*}

**Example:**

```
RP/0/RP0/CPU0:router(config-l2vpn-xc-mp2mp-ad-sig)# ce-id 10
```

Specifies the local Customer Edge Identifier.

**Step 12** Use the **commit** or **end** command.

**commit** - Saves the configuration changes and remains within the configuration session.

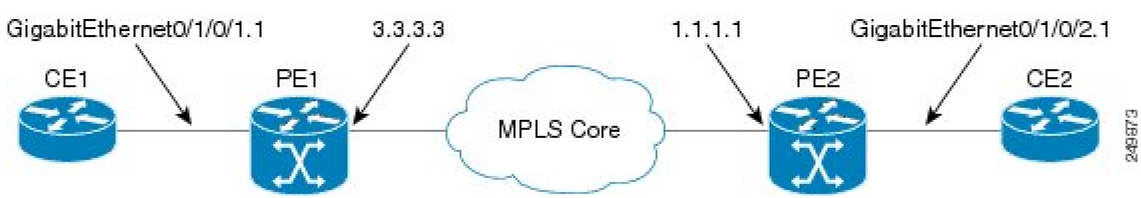
**end** - Prompts user to take one of these actions:

- **Yes** - Saves configuration changes and exits the configuration session.
- **No** - Exits the configuration session without committing the configuration changes.
- **Cancel** - Remains in the configuration mode, without committing the configuration changes.

# VPWS with BGP Autodiscovery and BGP Signaling

The following figure illustrates an example of configuring and verifyingVPWS with BGP autodiscovery (AD) and BGP Signaling.

***Figure 2: VPLS with BGP autodiscovery and BGP signaling***




### Configuration at PE1:

```
l2vpn
  xconnect group gr1
   mp2mp mp1
   vpn-id 100
   l2 encapsulation vlan
    autodiscovery bgp
    rd auto
    route-target 2.2.2.2:100
    ! Signaling attributes
```

```
    signaling-protocol bgp
    ce-id 1
    interface GigabitEthernet0/1/0/1.1 remote-ce-id 2
```

### Configuration at PE2:

```
l2vpn
  xconnect group gr1
   mp2mp mp1
   vpn-id 100
   l2 encapsulation vlan
    autodiscovery bgp
    rd auto
    route-target 2.2.2.2:100
    ! Signaling attributes
    signaling-protocol bgp
    ce-id 2
    interface GigabitEthernet0/1/0/2.1 remote-ce-id 1
```

### Verification:

**PE1:**

```
PE1# show l2vpn discovery xconnect

Service Type: VPWS, Connected

List of VPNs (1 VPNs):

XC Group: gr1, MP2MP mp1

   List of Local Edges (1 Edges):

     Local Edge ID: 1, Label Blocks (1 Blocks)

       Label base Offset   Size     Time Created

       ---------- ------   ----     -------------------

       16030     1         10       01/24/2009 21:23:04

        Status Vector: 9f ff

   List of Remote Edges (1 Edges):

     Remote Edge ID: 2, NLRIs (1 NLRIs)

       Label base Offset   Size     Peer ID         Time Created

       ---------- ------   ----     --------------- -------------------

      16045     1         10       1.1.1.1         01/24/2009 21:29:35

       Status Vector: 7f ff


PE1# show l2vpn xconnect mp2mp detail

Group gr1, MP2MP mp1, state: up

VPN ID: 100

VPN MTU: 1500
```

```
L2 Encapsulation: VLAN
Auto Discovery: BGP, state is Advertised (Service Connected)
     Route Distinguisher: (auto) 3.3.3.3:32770
   Import Route Targets:
       2.2.2.2:100
   Export Route Targets:
       2.2.2.2:100
   Signaling protocol:BGP
    CE Range:10
….
Group gr1, XC mp1.1:2, state is up; Interworking none
Local CE ID: 1, Remote CE ID: 2, Discovery State: Advertised
AC: GigabitEthernet0/1/0/1.1, state is up
   Type VLAN; Num Ranges: 1
   VLAN ranges: [1, 1]
   MTU 1500; XC ID 0x2000013; interworking none
PW: neighbor 1.1.1.1, PW ID 65538, state is up ( established )
   PW class not set, XC ID 0x2000013
   Encapsulation MPLS, Auto-discovered (BGP), protocol BGP
     MPLS         Local                          Remote
     ------------ ------------------------------ -----------------------------
     Label       16031                          16045
     MTU         1500                            1500
   Control word enabled                        enabled
     PW type     Ethernet VLAN                  Ethernet VLAN
     CE-ID       1                              2
     ------------ ------------------------------ -----------------------------

...
PE1# show bgp l2vpn vpws
BGP router identifier 3.3.3.3, local AS number 100
BGP generic scan interval 60 secs
```

```
BGP table state: Active
Table ID: 0x0
BGP main routing table version 913
BGP NSR converge version 3
BGP NSR converged
BGP scan interval 60 secs
Status codes: s suppressed, d damped, h history, * valid, > best
             i - internal, S stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network           Next Hop       Rcvd Label     Local Label
Route Distinguisher: 1.1.1.1:32775
*>i2:1/32             1.1.1.1         16045           nolabel
*>i3:1/32             1.1.1.1         16060           nolabel
Route Distinguisher: 3.3.3.3:32770 (default for vrf gr1:mp1)
*> 1:1/32             0.0.0.0         nolabel         16030
*>i2:1/32             1.1.1.1         16045           nolabel
*>i3:1/32             1.1.1.1         16060           nolabel

Processed 5 prefixes, 5 paths
```

**PE2:**

```
PE2# show l2vpn discovery xconnect
Service Type: VPWS, Connected
List of VPNs (1 VPNs):
XC Group: gr1, MP2MP mp1
   List of Local Edges (2 Edges):
     Local Edge ID: 2, Label Blocks (1 Blocks)
       Label base Offset   Size     Time Created
       ---------- ------   ----     -------------------
       16045     1         10       01/24/2009 21:09:14
       Status Vector: 7f ff
     Local Edge ID: 3, Label Blocks (1 Blocks)
       Label base Offset   Size     Time Created
```

```
        ---------- ------   ----     -------------------
        16060     1         10       01/24/2009 21:09:14
        Status Vector: 7f ff
    List of Remote Edges (1 Edges):
      Remote Edge ID: 1, NLRIs (1 NLRIs)
        Label base Offset   Size     Peer ID         Time Created
        ---------- ------   ----     --------------- -------------------
        16030     1         10       3.3.3.3         01/24/2009 21:09:16
      Status Vector: 9f ff

PE2# show l2vpn xconnect mp2mp detail
Group gr1, MP2MP mp1, state: up
VPN ID: 100
VPN MTU: 1500
L2 Encapsulation: VLAN
Auto Discovery: BGP, state is Advertised (Service Connected)
     Route Distinguisher: (auto) 1.1.1.1:32775
   Import Route Targets:
       2.2.2.2:100
   Export Route Targets:
       2.2.2.2:100
   Signaling protocol:BGP
     CE Range:10
...
Group gr1, XC mp1.2:1, state is up; Interworking none
Local CE ID: 2, Remote CE ID: 1, Discovery State: Advertised
AC: GigabitEthernet0/1/0/2.1, state is up
   Type VLAN; Num Ranges: 1
   VLAN ranges: [1, 1]
   MTU 1500; XC ID 0x2000008; interworking none
PW: neighbor 3.3.3.3, PW ID 131073, state is up ( established )
   PW class not set, XC ID 0x2000008
```

```
   Encapsulation MPLS, Auto-discovered (BGP), protocol BGP
     MPLS         Local                          Remote
     ------------ ------------------------------ -----------------------------
     Label        16045                          16031
     MTU          1500                            1500
     Control word enabled                         enabled
     PW type      Ethernet VLAN                   Ethernet VLAN
     CE-ID        2                              1
     ------------ ------------------------------ -----------------------------
...

PE2# show bgp l2vpn vpws
BGP router identifier 1.1.1.1, local AS number 100
BGP generic scan interval 60 secs
BGP table state: Active
Table ID: 0x0
BGP main routing table version 819
BGP NSR converge version 7
BGP NSR converged
BGP scan interval 60 secs
Status codes: s suppressed, d damped, h history, * valid, > best
             i - internal, S stale
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network            Next Hop       Rcvd Label     Local Label
Route Distinguisher: 1.1.1.1:32775 (default for vrf gr1:mp1)
*>i1:1/32             3.3.3.3         16030           nolabel
*> 2:1/32             0.0.0.0         nolabel         16045
*> 3:1/32             0.0.0.0         nolabel         16060
Route Distinguisher: 3.3.3.3:32770
*>i1:1/32             3.3.3.3        16030           nolabel

Processed 4 prefixes, 4 paths
```

CHAPTER 4

# BGP Dynamic Neighbors

Earlier, IOS-XR supported explicitly configured or static neighbor configuration. BGP dynamic neighbor support allows BGP peering to a group of remote neighbors that are defined by a range of IP addresses. Each range can be configured as a subnet IP address.

In larger BGP networks, implementing BGP dynamic neighbors can reduce the amount and complexity of CLI configuration and save CPU and memory usage. Both IPv4 and IPv6 peering are supported.

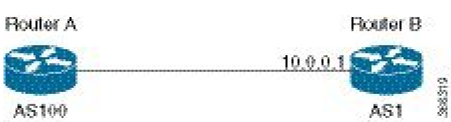


# Configuring BGP Dynamic Neighbors using Address Range

The existing neighbor command is extended to accept a prefix instead of an address.

In the following task, Router B is configured as a remote BGP peer. After a subnet range is configured, a TCP session is initiated by Router B which has an IP address in the subnet range and a new BGP neighbor is dynamically established.

After the initial configuration of subnet ranges and activation of the peer neighbor, dynamic BGP neighbor creation does not require any further CLI configuration on the Router A.




### Configuration

```
Router# configure
Router(config)# router bgp as-number
Router(config-bgp)# neighbor address prefix
Router(config-bgp-nbr)# remote-as as-number
Router(config-bgp-nbr)# update-source interface
Router(config-bgp-nbr)# address-family ipv4 unicast
Router# commit
```

### Running Configuration

```
Router# show running-config router bgp

router bgp 100
address-family ipv4 unicast
```

```
!
neighbor 12.12.12.0/24
  remote-as 100
  update-source TenGigE0/11/0/5
  address-family ipv4 unicast
  !
!
!
```

# Configuring BGP Dynamic Neighbors Using Address Range With Authentication

The following task shows how to configure BGP dynamic neighbors using address range with Message Digest 5 (MD5) authentication.

```
Router# configure
Router(config)# router bgp as-number
Router(config-bgp)# neighbor address prefix
Router(config-bgp-nbr)# remote-as as-number
Router(config-bgp-nbr)# password {clear | encrypted} password
Router(config-bgp-nbr)# update-source interface
Router(config-bgp-nbr)# address-family ipv4 unicast
Router# commit
```

### Running Configuration

```
Router# show running-config router bgp

router bgp 100
address-family ipv4 unicast
!
neighbor 12.12.12.0/24
  remote-as 100
  password encrypted 053816063349401D
  update-source TenGigE0/11/0/5
  address-family ipv4 unicast
  !
!
!
```

### Configuring EA Authentication

The following task shows how to configure the EA authentication.

**Note** Configuring EA authentication is a prerequisite for configuring BGP dynamic neighbors with EA authentication.

```
RP/0/RP0/CPU0:R1(config)# key chain bgp_ea
RP/0/RP0/CPU0:R1(config-bgp_ea)# key 1
RP/0/RP0/CPU0:R1(config-bgp_ea-1)# accept-lifetime 00:00:00 january 01 2019 infinite
RP/0/RP0/CPU0:R1(config-bgp_ea-1)# key-string bgp_ea_key
RP/0/RP0/CPU0:R1(config-bgp_ea-1)# send-lifetime 00:00:00 january 01 2019 infinite
RP/0/RP0/CPU0:R1(config-bgp_ea-1)# cryptographic-algorithm HMAC-SHA1-12
RP/0/RP0/CPU0:R1(config-bgp_ea-1)# root
RP/0/RP0/CPU0:R1(config)# commit
Sat Sep  5 10:30:37.219 UTC
RP/0/RP0/CPU0:R1(config)# show run key chain
```

```
Sat Sep  5 10:30:41.976 UTC
key chain bgp_ea
 key 1
  accept-lifetime 00:00:00 january 01 2019 infinite
  key-string password 01110114640E07302A4957
  send-lifetime 00:00:00 january 01 2019 infinite
  cryptographic-algorithm HMAC-SHA1-12
 !
!
RP/0/RP0/CPU0:R1# show install active summary
Sat Sep  5 10:36:51.537 UTC
Active Packages:    XR: 115    All: 1080
Label:              7.0.1

Optional Packages                                                    Version
---------------------------------------------------- ---------------------------
xr-bgp                                                         7.0.1v1.0.0-1
```

The following task shows how to configure BGP dynamic neighbors using address range with EA authentication.

```
Router# configure
Router(config)# router bgp as-number
Router(config-bgp)# neighbor address prefix
Router(config-bgp-nbr)# remote-as as-number
Router(config-bgp-nbr)# keychain bgp_ea
Router(config-bgp-nbr)# address-family ipv4 unicast
Router(config-bgp-nbr)# route-policy name
Router(config-bgp-nbr)# route-policy name
Router# commit
```

### Running Configuration

```
router bgp 100
neighbor 6.1.1.2
  remote-as 200
  keychain bgp_ea
  address-family ipv4 unicast
   route-policy bgp_policy in
   route-policy bgp_policy out
  !
```

# Maximum-peers and Idle-watch timeout

In the following task, **maximum-peers** and **idle-watch timeout** commands are configured for a remote BGP peer.

### Configuration

```
Router# configure
Router(config)# router bgp as-number
Router(config-bgp)# neighbor address prefix
Router(config-bgp-nbr)# remote-as as-number
Router(config-bgp-nbr)# password {clear | encrypted} password
Router(config-bgp-nbr)# maximum-peers number
Router(config-bgp-nbr)# update-source interface
Router(config-bgp-nbr)# idle-watch-time number
Router(config-bgp-nbr)# address-family ipv4 unicast
Router# commit
```

### Running Configuration

```
Router# show running-config router bgp
router bgp 100
address-family ipv4 unicast
!
neighbor 12.12.12.0/24
  remote-as 100
  password encrypted 053816063349401D
  maximum-peers 10
  update-source TenGigE0/11/0/5
  idle-watch-time 40
  address-family ipv4 unicast
  !
!
!
```

CHAPTER 5

# BGP Prefix Independent Convergence

*Table 6: Feature History Table*

| Feature Name | Release Information | Feature Description |
|---|---|---|
| BGP PIC Backup Path when the Primary Path is a Static Route with the next hop as an IP Address. | Release 7.5.1 | This feature is now supported on routers that operate in native and compatibiltiy mode.<br>This feature enables BGP PIC backup path when the primary path is a static route with the next hop as an IP Address. |

**Restrictions:**

- Ensure that the Border Gateway Protocol (BGP) and the IP or Multiprotocol Label Switching (MPLS) network is up and running at the customer site that is connected to the provider site by more than one path (multihomed).
- BGP PIC does not support instances where the sum of number of primary paths and backup paths is greater than 2. Hence, only one primary path and one backup path are supported.
- Ensure that the backup or alternate path has a unique next hop that is not the same as the next hop of the best path.

## BGP PIC: Export of Backup Path Agnostic to its Multipath Eligibility

The BGP PIC: Export of Backup Path Agnostic to its Multipath Eligibility feature improves BGP convergence after a network failure. This convergence applies to both core and edge failures. The BGP PIC pre-programs a backup path so that when a failure is detected, the backup path can immediately take over, thus enabling fast failover. This feature enables BGP PIC on VPNv4 with additional paths or when the multiple paths that are ineligible to be multipath are received from the same neighbor. For backup paths to be multipath eligible, all the following attributes in the backup paths must be the same: weight, local preference, autonomous system path, origin code, Multi Exit Discriminator (MED), and Interior Gateway Protocol (iGP) distance. Also, the next hop router for each multipath must be different. This feature introduces flexibility to allow the import of backup paths to the VRF even if the said attributes are not the same.

### Configuration Example

```
Router# router bgp 10
Router(config-bgp)# address-family vpnv4 unicast
```

```
router(config-bgp-af)# export to vrf allow backup
```

### Running Configuration

```
router bgp 10
 address-family vpnv4 unicast
  export to vrf allow backup
```


- BGP PIC Implementation Considerations, on page 166
- Configure BGP PIC, on page 166


# BGP PIC Implementation Considerations

- BGP PIC over BVI (core or edge) is not supported.
- For labelled BGP loopback peering, the system supports only one primary and one backup path. No support for BGP PIC multipath protect.
- PIC EDGE is supported for all services, such as IPv4, IPv6, VPNv4, VPNv6, 6PE, 6VPE, VPWS, VPLS, and EVPN, over labelled unicast address-family.

# Configure BGP PIC

### Procedure

**Step 1** **cef encap-sharing disable**

**Example:**

```
RP/0/RP0/CPU0:router(config)# cef encap-sharing disable
```

By default, without primary and backup path installation in the hardware, IPv4, IPv6, 6PE (per-vrf), 6VPE (per-vrf/per-ce), L3VPN (per-vrf/per-ce) has good convergence.

When the mode is a per-prefix by default, BGP-PIC does not give good convergence, hence you must do hardware-assisted PIC. For this, configure the **cef encap-sharing disable** command in XR Config mode.

With hardware-assisted BGP PIC that is configured using the **cef encap-sharing disable** command, separate hardware resources (FEC/EEDB) are allocated for every prefix. Cisco recommends you to make sure that the router has sufficient hardware resources for the resource allocation.

**Caution** This CLI reprograms the CEF completely and impacts traffic. We recommend that you do it in the maintenance window.

**Note** The **cef encap-sharing disable** command does not take effect in the SRv6 core.

**Step 2** **router bgp** *as-number*

**Example:**

```
RP/0/RP0/CPU0:router(config)# router bgp 100
```

Specifies the autonomous system number and enters the BGP configuration mode, allowing you to configure the BGP routing process.

**Step 3** **address-family** {**vpnv4 unicast** | **vpnv6 unicast** | **ipv4 unicast** | **ipv6 unicast**}

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp)# address-family ipv4 unicast
```

```
address-family ipv4 unicast
  additional-paths receive
  additional-paths selection route-policy backup 1
  allocate-label all
!
```

**Step 4** **additional-paths selection route-policy** *route-policy-name*

**Example:**

```
RP/0/RP0/CPU0:router(config-bgp-af)# additional-paths selection route-policy ap1
```

Configures extra paths selection mode for a prefix.

**Note** Use the **additional-paths selection** command with an appropriate route-policy to calculate backup paths and to enable Prefix-Independent Convergence (PIC) functionality.

The route-policy configuration is a prerequisite for configuring the additional-paths selection mode for a prefix. This is an example route-policy configuration to use with additional-selection command:

```
route-policy ap1
    set path-selection backup 1 install
  end-policy
```

CHAPTER 6

# Implementing Master Key Tuple Configuration

This feature specifies TCP Authentication Option (TCP-AO), which replaces the TCP MD5 option. TCP-AO uses the Message Authentication Codes (MACs), which provides the following:



## Master Key Tuple Configuration

This feature specifies TCP Authentication Option (TCP-AO), which replaces the TCP MD5 option. TCP-AO uses the Message Authentication Codes (MACs), which provides the following:

- Protection against replays for long-lived TCP connections
- More details on the security association with TCP connections than TCP MD5
- A larger set of MACs with minimal other system and operational changes

TCP-AO is compatible with Master Key Tuple (MKT) configuration. TCP-AO also protects connections when using the same MKT across repeated instances of a connection. TCP-AO protects the connections by using traffic key that are derived from the MKT, and then coordinates changes between the endpoints.

**Note** TCPAO and TCP MD5 are never permitted to be used simultaneously. TCP-AO supports IPv6, and is fully compatible with the proposed requirements for the replacement of TCP MD5.

Cisco provides the MKT configuration via the following configurations:

- keychain configuration
- tcp ao keychain configuration

The system translates each key, such “key_id” that is under a keychain, as MKT. The keychain configuration owns part of the configuration like secret, lifetimes, and algorithms. While the “tcp ao keychain” mode owns the TCP AO-specific configuration for an MKT (send_id and receive_id).

# Keychain Configurations

## Configuration Guidelines

In order to run a successful configuration, ensure that you follow the configuration guidelines:

- An allowed value range for both Send_ID and Receive_ID is 0 to 255.
- You can link only one keychain to an application neighbor.
- Under the same keychain, if you configure the same send_id key again under the keys that have an overlapping lifetime, then the old key becomes unusable until you correct the configuration.
- The system sends a warning message in the following scenarios:
  - If there is a change in Send_ID or Receive_ID.
  - If the corresponding key is currently active, and is in use by some connection.
- BGP neighbor can ONLY use one of the authentication options:
  - MD5
  - EA
  - AO

**Note** If you configure one of these options, the system rejects the other authentication options during the configuration time.

## Configuration Guidelines for TCP AO BGP Neighbor

The configuration guidelines are:

- Configure all the necessary configurations (key_string, MAC_algorithm, send_lifetime, accept_lifetime, send_id, receive_id) under key_id with the desired lifetime it wants to use the key_id for.
- Configure a matching MKT in the peer side with exactly same lifetime.
- Once a keychain-key is linked to tcp-ao, do not change the components of the key. If you want TCP to consider another key for use, you can configure that dynamically. Based on the ‘start-time’of send lifetime, TCP AO uses the key.
- Send_ID and Receive_ID under a key_id (under a keychain) must have the same lifetime range. For example, send-lifetime==accept-lifetime.

  TCP considers only expiry of send-lifetime to transition to next active key and it does not consider accept-lifetime at all.
- Do not configure a key with send-lifetime that is covered by another key’s send-lifetime.

For example, if there is a key that is already configured with send-lifetime of "04:00:00 November 01, 2017 07:00:00 November 01, 2017" and the user now configures another key with send-lifetime of "05:00:00 November 01, 2017 06:00:00 November 01, 2017", this might result into connection flap.

TCP AO tries to transition back to the old key once the new key is expired. However, if the new key has already expired, TCP AO can't use it, which might result in segment loss and hence connection flap.

- Configure minimum of 15 minutes of overlapping time between the two overlapping keys. When a key expires, TCP does not use it and hence out-of-order segments with that key are dropped.
- We recommend configuring send_id and receive_id to be same for a key_id for simplicity.
- TCP does not have any restriction on the number of keychains and keys under a keychain. The system does not support more than 4000 keychains, any number higher than 4000 might result in unexpected behaviors.

# Keychain Configuration

```
key chain <keychain_name>
   key <key_id>
      accept-lifetime <start-time> <end-time>
      key-string <master-key>
      send-lifetime <start-time> <end-time>
      cryptographic-algorithm <algorithm>
   !
!
```

# TCP Configuration

TCP provides a new tcp ao submode that specifies SendID and ReceiveID per key_id per keychain.

```
tcp ao
    keychain <keychain_name1>
        key-id <key_id> send_id <0-255> receive_id <0-255>
        !
```

Example:

```
tcp ao
 keychain bgp_ao
  key 0 SendID 0 ReceiveID 0
  key 1 SendID 1 ReceiveID 1
  key 2 SendID 3 ReceiveID 4
 !
 keychain ldp_ao
  key 1 SendID 100 ReceiveID 200
  key 120 SendID 1 ReceiveID 1
 !
```

# BGP Configurations

Applications like BGP provide the tcp-ao keychain and related information that it uses per neighbor. Following are the optional configurations per tcp-ao keychain:

- include-tcp-options
- accept-non-ao-connections

```
router bgp <AS-number>
neighbor <neighbor-ip>
  remote-as <remote-as-number>
  ao <keychain-name> include-tcp-options enable/disable  <accept-ao-mismatch-connections>
!
```

# XML Configurations

### BGP XML

### TCP-AO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Request>
 <Set>
  <Configuration>
   <IP_TCP>
    <AO>
      <Enable>
       true
      </Enable>
      <KeychainTable>
        <Keychain>
         <Naming>
          <Name> bgp_ao_xml </Name>
         </Naming>
         <Enable>
          true
         </Enable>
          <KeyTable>
           <Key>
            <Naming>
             <KeyID> 0 </KeyID>
            </Naming>
             <SendID> 0 </SendID>
             <ReceiveID> 0 </ReceiveID>
           </Key>
          </KeyTable>
        </Keychain>
      </KeychainTable>
    </AO>
   </IP_TCP>
  </Configuration>
 </Set>
 <Commit/>
</Request>
```

# Verification

To verify the keychain database, use the show tcp authentication keychain <keychain-name> command in EXEC mode. The following output displays all the keychain database details:

```
Keychain name: tcp_ao_keychain1, configured for tcp-ao
Desired key: 1
Detail of last notification from keychain:
Time: 'Jan 23 12:07:39.128', event: Config update, attr: Crypto algorithm, key: 1
Total number of keys: 1
Key details:
    Key ID: 1, Active, Valid
```

```
    Active_state: 1, invalid_bits: 0x0, state: 0x110
    Key is configured for tcp-ao, Send ID: 1, Receive ID: 1
    Crypto algorithm: AES_128_CMAC_96, key string chksum: 00028222
    Detail of last notification from keychain:
    Time: 'Jan 23 12:07:39.128', event: Config update, attr: Crypto algorithm
    No valid overlapping key
    No keys invalidated

Total number of usable (Active & Valid) keys: 1
    Keys: 1,
Total number of peers: 24
Peer details:
    Peer: 0x7fc2f00242f8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc2f0024618,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc2f00247f8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc2f00249d8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc2f0024bb8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc320037a08,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc320037d78,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc3200386d8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc3200388b8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc320038a98,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

    Peer: 0x7fc35000d3f8,
    Current key: 1
```

```
Traffic keys: send_non_SYN: 00476017, recv_non_SYN: ffd520f9
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:41.953, reason: Peer requested rollover

Peer: 0x7fc320038e78,
Current key not yet available
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

Peer: 0x7fc350012758,
Current key not yet available
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

Peer: 0x7fc2f0026bc8,
Current key not yet available
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

Peer: 0x7fc320048b08,
Current key: 1
Traffic keys: send_non_SYN: 004a05b5, recv_non_SYN: fff639b2
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:44.209, reason: No current key set

Peer: 0x7fc2f4008388,
Current key: 1
Traffic keys: send_non_SYN: 0029837c, recv_non_SYN: 002af030
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:44.229, reason: No current key set

Peer: 0x7fc350017198,
Current key: 1
Traffic keys: send_non_SYN: ffdb7322, recv_non_SYN: fff1fb23
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:45.419, reason: Peer requested rollover

Peer: 0x7fc320049098,
Current key: 1
Traffic keys: send_non_SYN: ffed0d67, recv_non_SYN: ffe4f959
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:55.180, reason: No current key set

Peer: 0x7fc32005d2a8,
Current key: 1
Traffic keys: send_non_SYN: 0021b461, recv_non_SYN: fffe679e
RNext key: 1
Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
Last 1 keys used:
    key: 1, time: Jan 23 12:07:56.894, reason: No current key set

Peer: 0x7fc350035c88,
Current key: 1
Traffic keys: send_non_SYN: 00296167, recv_non_SYN: fff1c236
```

```
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
    Last 1 keys used:
        key: 1, time: Jan 23 12:07:57.859, reason: Peer requested rollover

    Peer: 0x7fc35003fb18,
    Current key: 1
    Traffic keys: send_non_SYN: ffc95844, recv_non_SYN: ffcdfd4f
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
    Last 1 keys used:
        key: 1, time: Jan 23 12:08:00.754, reason: Peer requested rollover

    Peer: 0x7fc350049638,
    Current key: 1
    Traffic keys: send_non_SYN: 002ff48b, recv_non_SYN: ffbe71b9
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
    Last 1 keys used:
        key: 1, time: Jan 23 12:08:10.014, reason: Peer requested rollover

    Peer: 0x7fc350053928,
    Current key: 1
    Traffic keys: send_non_SYN: 00206914, recv_non_SYN: 001df9bc
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000
    Last 1 keys used:
        key: 1, time: Jan 23 12:08:12.422, reason: Peer requested rollover

    Peer: 0x7fc2f401f3b8,
    Current key not yet available
    RNext key: 1
    Traffic keys: send_non_SYN: 00000000, recv_non_SYN: 00000000

Total number of Send IDs: 1
Send ID details:
    SendID: 1, Total number of keys: 1
        Keys: 1,
Total number of Receive IDs: 1
Receive ID details:
    ReceiveID: 1, Total number of keys: 1
        Keys: 1,

RP/0/RP0/CPU0:stoat#
```