# ZebOS-XP®
# Network Platform

## Version 1.4
## Extended Performance

## Border Gateway Protocol
## Developer Guide

December 2015

IP Infusion Inc.
3965 Freedom Circle, Suite 200
Santa Clara, CA 95054
+1 408-400-1900
http://www.ipinfusion.com/

For support, questions, or comments via E-mail, contact:
support@ipinfusion.com

# Contents

# Preface

This guide describes the ZebOS-XP application programming interface (API) for Border Gateway Protocol (BGP).

## Audience

This guide is intended for developers who write code to customize and extend BGP.

## Conventions

Table P-1 shows the conventions used in this guide.

**Table P-1: Conventions**

| Convention | Description |
|---|---|
| *Italics* | Emphasized terms; titles of books |
| Note: | Special instructions, suggestions, or warnings |
| `monospaced type` | Code elements such as commands, functions, parameters, files, and directories |

## Contents

This document contains these chapters and appendices:

## Related Documents

The following guides are related to this document:

- *Border Gateway Protocol Command Reference*
- *Open Shortest Path First Command Reference*
- Unicast Configuration Guide
- *Installation Guide*
- *Architecture Guide*

Note: All ZebOS-XP technical manuals are available to licensed customers at http://www.ipinfusion.com/support/document_list.

## Support

For support-related questions, contact support@ipinfusion.com.

## Comments

If you have comments, or need to report a problem with the content, contact techpubs@ipinfusion.com.

# CHAPTER 1 ZebOS-XP BGP Architecture Overview

## What is BGP?

Border Gateway Protocol (BGP) is the routing protocol for the Internet. BGP shares routing information between autonomous systems (ASs). An AS is comprised of devices under the same administrative control. An AS can use a collection of Interior Gateway Protocols (IGPs) to share routing information within the same administrative domain.

ZebOS-XP implements BGP4 for IPv4 and IPv6 and supports classless inter-domain routing (CIDR) and route aggregation. During BGP neighbor establishment, TCP builds the adjacency. There is no requirement to implement retransmission, acknowledgement, and sequencing, because TCP handles these features. However, BGP maintains its own keepalive mechanism for faster detection of peer availability.

BGP is typically an inter-AS routing protocol. The intra-AS routing is usually handled by the IGP (for example, OSPF or RIP). BGP requires a route to be reachable before it can advertise it to another AS. The routes are advertised to the other ASs through network layer reachability information.

Figure 1-1 shows how BGP works with the Internet.

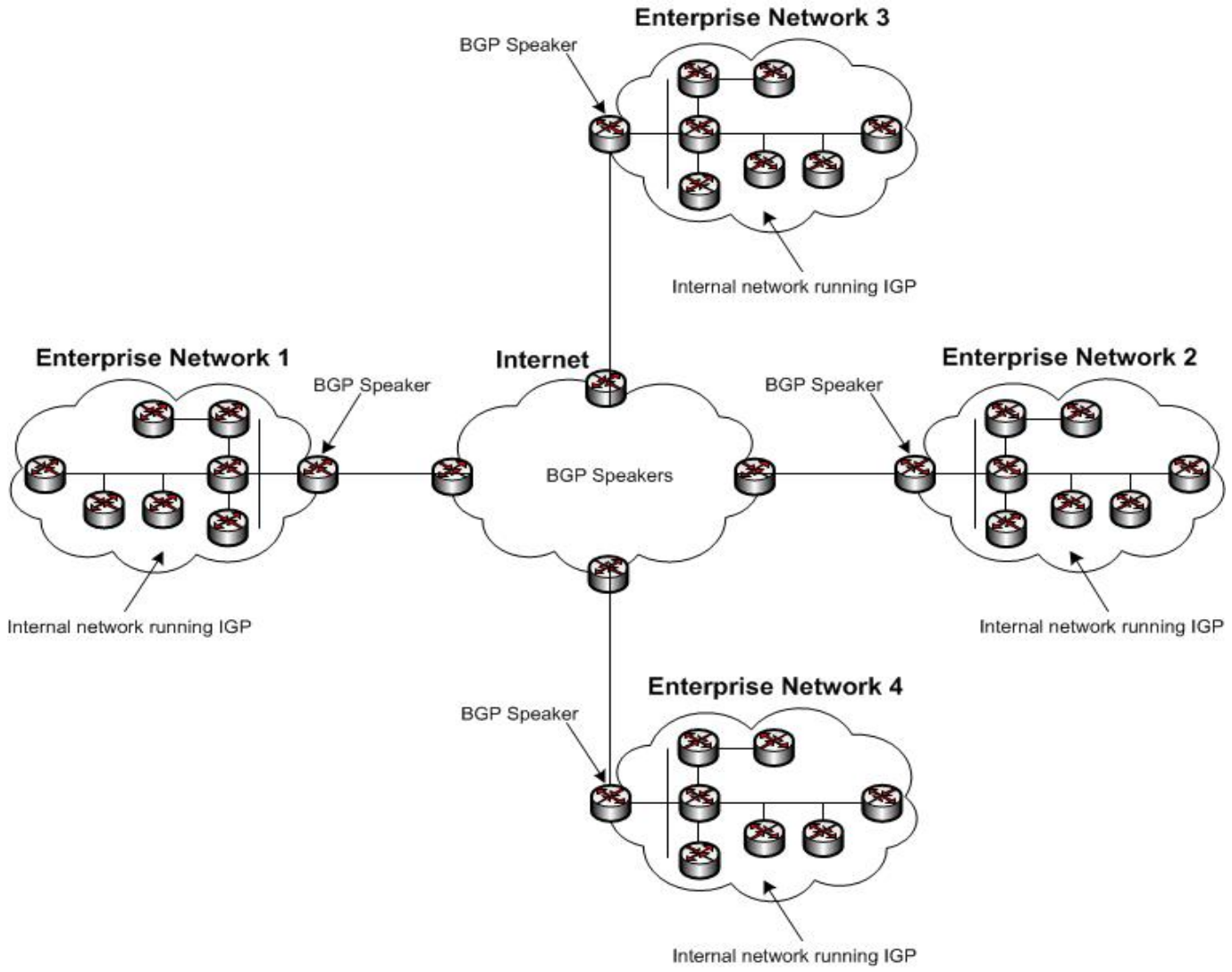



**Figure 1-1: BGP In the Internet**

## eBGP and iBGP

There are two types of BGP:

- Exterior BGP (eBGP) exchanges routing information between different ASs.
- Interior BGP (iBGP) exchanges routing information within the same AS. In iBGP, all routers running iBGP should peer with one another which implies that iBGP routers should have full-mesh connectivity. This is a limitation of iBGP implementation; however, there are methods to overcome this limitation.

# BGP Messages

There are four important BGP message types:

- OPEN
- UPDATE
- NOTIFICATION
- KEEPALIVE

### OPEN

An OPEN message from the BGP speaker is sent after a TCP connection is formed between the peers. This message contains the version, autonomous system number (ASN), hold time, BGP identifier, length of optional parameters, and optional parameters.

### UPDATE

UPDATE messages are exchanged between the BGP speakers, once the two BGP speakers have established neighbor adjacency. UPDATE messages typically contain the routes that must be advertised to the corresponding peer. It also contains routes that must be removed from the routing table. Hence, the UPDATE message controls the routing information that goes into the routing table.

### NOTIFICATION

When an error occurs in a BGP session, and the session must be torn down, NOTIFICATION messages are sent from the corresponding peer. Once the other peer receives the message, the TCP connection is torn down, and the speakers are no longer peers with each other.

### KEEPALIVE

This message serves as the connectivity check between BGP speakers. The value of this KEEPALIVE timer highly depends on the hold time advertised in the OPEN message. BGP requires that the KEEPALIVE time is one third of the hold time, that is, hold time is three times the KEEPALIVE time.

# When Should BGP Be Used?

BGP should be used when:

- Policies must be configured on ingress/egress interfaces, for example, the AS in question must route traffic from one AS to another AS, through itself.
- Redundancy is required by peering with multiple Internet service providers.

BGP is not required, or should not be used if:

- No policies are required to be implemented.
- The network does not have enough bandwidth, router does not have enough memory, and CPU does not have fast enough processing speed.

## BGP Multipath

BGP multipath feature is useful for load-balancing on forwarding path when multiple equal paths are available for a given prefix. The feature is available for both eBGP and iBGP. BGP Multipath allows installation into the IP routing table of multiple BGP paths to the same destination. These paths are installed in the table together with the best path for load sharing. BGP Multipath does not affect best path selection. For example, a router still designates one of the paths as the best path, according to the algorithm, and advertises this best path to its neighbors.

For information regarding BGP multipath functions, see Chapter 19, *BGP Command API*.

# CHAPTER 2 BGP Features

An overview of the features supported in ZebOS-XP BGP is provided in this chapter.

## RFC 4271 Support

ZebOS-XP BGP complies with RFC 4271 and supports the following:

- Missing Multi-Exit Discriminator (MED). If the MED attribute is missing in a received Update message, ZebOS-XP takes the default value of the MED while updating the database.
- The Hold timer can be configured on a per-connection basis
- A BGP speaker can be configured to prepend its autonomous system (AS) number multiple times in the AS-PATH
- Atomic Aggregate processing
- Next-hop calculation. The Next-hop attribute defines the IP address of the router to be used as the next hop to the destinations listed in the Update message
- Cease notification. In the absence of any fatal errors, a BGP peer can close its BGP connection by sending a Cease Notification message
- Finite State Machine (FSM) error handling. Any error detected by the BGP FSM is indicated by sending the FSM notification message
- Route selection per Section 9.1.2
- Tie-breaking procedure per Section 9.1.2.2
- Overlapping routes per Section 9.1.4
- Aggregating route information per Section 9.2.2.2
- BGP MD5 verification. Protects BGP sessions via the TCP MD5 signature.

## RFC 4760 Support

Partitioning of Subsequent Address Family Identifier (SI) space is supported per RFC 4760. To accommodate 6PE and 6VPE, the IPV6-SI-MPLS_LABEL and IPV6-SI_BGP_MPLS_VPN I-SI combinations are supported.

## RFC 4273 Support

The BGP-4 MIB is per RFC 4273 to define BGP Notifications, instead of BGP Traps, and define new notification types.

## Route Refresh for IPv4 and IPv6

This feature requests that a peer re-advertise all of the prefixes learned from other peers (its Adj-RIB-Out).

## BGP Community Attribute

This dynamically influences the routing decisions of ASs: in a topology of two ASs, AS1 and AS2, AS1 can advertise a route to AS2, and simultaneously provide dynamic instructions that AS1 is not to advertise the route externally. In this instance, the Community attribute assigned to the advertised route is `no-export`.

## BGP Extended Communities Attribute

This transitive optional BGP attribute consists of a set of extended communities. All routes with the Extended Communities attribute belong to the communities listed in the attribute.

The Extended Communities attribute provides two important enhancements over the existing BGP Community attribute:

- It provides an extended range, ensuring that communities can be assigned for myriad uses, without overlap.
- The addition of a `type` field that provides structure for the community space.

It allows the usage of policy based on the application for which the community value is used.

Also supported are the Opaque extended community type, and four-byte AS specific extended community attribute for BGP speakers that use four-byte AS numbers.

## Four-Byte ASN

BGP can be configured to encode/decode the AS number (ASN) in four-byte format to support a higher number of ASs.

## BGP Confederation

BGP confederation is a way to handle the explosion of an internal BGP (iBGP) mesh within an AS. IP Infusion Inc. recommends using this only for cases that involve a large number of iBGP peering sessions per router.

AS path checks and error handling are per RFC 5065. As such, updates containing AS-CONFED SET or AS_CONFED_SEQUENCE are not transmitted to peers that are not members of the local confederation.

## BGP Next-hop Tracking

Next-hop Tracking (NHT) reduces BGP route convergence time when IGP routes are changed and improves BGP CPU utilization. BGP is notified asynchronously, by NSM, of IGP route changes, letting users configure the next-hop delay time. This feature is turned off by default.

## Graceful Restart and Graceful Reset

These features help to enable smooth restarts and provide more options for recovery. BGP graceful restart functionalities are available for IPv4, IPv6 and IPv4 BGP-MPLS-VPN address families.

Graceful Restart makes it possible for a router to restart, and re-join its peers in the network, without rebuilding its routing database.

## Graceful Shutdown

The ZebOS-XP BGP graceful shutdown feature is based on IETF draft "draft-ietf-grow-bgp-gshut-03", Graceful BGP Session Shutdown.

Note: ZebOS-XP does not support graceful shutdown for iBGP sessions.

Without the use of the BGP graceful shutdown feature, when an autonomous system border router (ASBR) with an eBGP peering session with another ASBR is shut down for maintenance, the router removes the routes and sets off the BGP convergence on its BGP peers by withdrawing its routes. The results are packet loss due to the BGP routers being unreachable during the convergence. The BGP graceful shutdown feature allows for a planned shutdown of ASBRs to take place with minimal or no loss of traffic. Packet loss is minimized by keeping the route to the affected router undisturbed even after shutdown. This is achieved by lowering the local preference value (LOCAL_PREF attribute) of a BGP router to be shut down so that the other routers residing in the same autonomous system can still use this route during the convergence until these routers become aware of alternative routes.

The local preference value is used to indicate the preferred route when there are multiple routes to the same destination in a single routing database. The route having a lower preference value is used to indicate to the local autonomous system that this route is not preferred. The route that is propagated to all of the routers and access servers in the local autonomous system is the route with the highest local preference value. Using the local preference value, the BGP graceful shutdown feature gives the BGP routers a valid route to use during BGP convergence.

## Route Server, Filtering, and Load Balancing

These capabilities enable flexibility in configuration and control. BGP route server is used at the Internet junction points for distributing routes to other BGP speakers, based on policies between the eBGP speakers. ZebOS-XP route server is capable of handling BGP-MPLS-VPN routes.

## Inter AS BGP MPLS VPNs

Provider Edge (PE) routers in different ASs can communicate with each other and exchange routing information, including:

- eBGP redistribution of labeled VPN IPv4/IPv6 routes from an AS to the neighboring AS
- Multihop eBGP redistribution of labeled VPN IPv4/IPv6 routes between source and destination ASs, with eBGP redistribution of labeled IPv4/IPv6 routes from an AS to the neighboring AS.

## BGP MPLS for IPv4 VPN

ZebOS-XP BGP VPN is a fully scalable implementation of IETF 2547-bis. ZebOS-XP BGP can be used either as a PE/P node in the Service provider network, or as CPE to connect to the Service provider edge using eBGP. Other ZebOS-XP routing protocols, such as RIP and OSPF, are also extended to work as Customer Edge–Provider Edge (CE-PE) routing protocols.

ZebOS-XP BGP uses multiple forwarding tables to provide network Isolation of VPN traffic inside the service provider network. A separate default forwarding table can be used to contain public Internet routes and site-based VRF tables for VPN routes. The PE router maintains a VRF table, per site, that shares the same routes. When an IP packet is received on a PE-CE interface, the corresponding VRF is used for the destination IP address lookup.

### CE-PE Interaction

The CE devices learn and advertise routes to other sites in its VPNs. ZebOS-XP BGP, OSPF, and RIP can be used for CE-PE route exchange. BGP in PE installs the routes learned in the corresponding VRF table of the site, based on the incoming interface.

### PE-PE Interaction

At the PE device, ZebOS-XP BGP installs the VPN routes that were learned from the attached CE in VRF tables. ZebOS-XP BGP uses Multiprotocol BGP (MP-BGP) to distribute VPN routes to other PE BGPs across the SP network. Routes that are exported to BGP from VRF tables are converted into unique addresses in the VPN-IP address family using route distinguishers (RDs). The VPN-IP route is a 12-byte quantity that consists of an RD and the IPv4 address. ZebOS-XP BGP uses Route Targets to advertise by PE devices with the Router Target attribute. Every VRF is associated with a set of route targets. The import route target list specifies the routes that can be installed in a VRF. Routes advertised to other PE devices are tagged with the export route targets.

## BGP MPLS for IPv6 VPN

The BGP MPLS IP VPN extension for IPv6 VPN provides the same functionality as BGP MPLS for IPv4 VPN, but extends this capability for IPv6 networks.

In addition, ZebOS-XP BGP supports PEs in multi-AS backbones, per RFC 4659, Section 8.

## IPv6 Islands over IPv4 MPLS Using 6PE

IPv6 Provider Edge routers (6PE) interconnect IPv6 islands over an IPv4 MPLS cloud, so that dynamically established IPv4 signaled MPLS Label Switched Paths (LSPs) can be used without explicit tunnel configuration.

## Capability Negotiation

This feature introduces an optional parameter, called Capabilities. It facilitates the introduction of new features in BGP by providing graceful capability negotiation, without requiring the BGP peers to be terminated.

## Outbound Routing Filter

The Outbound Routing Filter (ORF) exchanges filtering data among routing peers, that each peer implements on behalf of the others, to block certain routing updates. This feature is advantageous because 1) local BGP speakers consume less resources, 2) less work is required for remote BGP speakers, 3) less link bandwidth usage is required, 4) and it provides the ability to configure many neighboring routers from a central route reflector.

## BGP Multiple Instance

The ZebOS-XP BGP Multiple Instance feature allows two or more independent routing instances to exist inside a single BGP process or task.

## MBGP Support

Multicast BGP (MBGP) is a set of extensions to BGP-4, allowing Internet multicast routing policy both within and between BGP autonomous systems. BGP carries two sets of routes when using MBGP: one for unicast routing, and the other for multicast routing. Protocol Independent Multicast (PIM) uses the routes associated with multicast routing to build the data distribution trees necessary to route the multicast traffic. The benefits of MBGP include:

- A network can support unicast and multicast topologies.
- A network can carry routing information for multiple network layer protocol address families.
- All routing policy capabilities of BGP can be applied to MBGP.
- All BGP commands can be used with MBGP.

The main restriction of MBGP is that MBGP routes cannot be redistributed into BGP

## Route Aggregation

Aggregation is the process of combining the characteristics of several different routes, so that a single route can be advertised. Aggregation reduces the amount of information a BGP speaker can store and exchange with other speakers. It can be part of the decision process to reduce the amount of routing information that will be placed in the Adj-RIBs-Out. The general rule of route aggregation is that routes are aggregated only if the corresponding attributes of the routes are identical.

## Route Reflection Support

According to BGP regulations, iBGP Speaker A, cannot advertise a route learned from iBGP Speaker B to iBGP Speaker C. As a result of this constraint, an AS with hundreds of routing nodes can become a serious management problem. Route reflection is used to correct this problem. In an AS, multiple BGP routers can peer with a central server and the route reflector; then, route reflectors can peer with one another. The BGP routers peered with the reflector exchange routing information with it, and it, in turn, pass (or reflect) the information between clients, and to other iBGP and External BGP (eBGP) peers. Route reflection is very useful and is usually recommended for ASs in which a fairly large number of BGP sessions must be built between routers.

## Route Flap Dampening

Route flapping is a symptom of route instability. A route in the routing table may disappear and reappear intermittently. This is known as flapping. It occurs when BGP sends a routing table, then withdraws it. A route that appears and disappears intermittently and repeatedly, propagates BGP UPDATE and WITHDRAWN messages onto the Internet. This excessive flow of traffic can use up bandwidth, and drive up CPU utilization of routers.

Route Flap Dampening categorizes routes as either well-behaved or ill-behaved. A well-behaved route shows a high degree of stability during an extended period of time (as determined by configurable limits). Ill-behaved routes are no longer advertised, but suppressed until there is an indication that the route has become stable. With route dampening, each time a route flaps, it is given a penalty. Whenever the penalty reaches a predefined limit, the route is suppressed; and while suppressed, it can still accumulate penalties. Unsuppressing it involves a similar mode of operation, in which the penalty value is incrementally reduced, until the reuse limit is reached, and the route can be advertised again.

## Route Map Support

This is a control mechanism. Route maps are used with BGP to control and modify routing information, and define the conditions by which routes are redistributed between routers and routing processes.

## BGP - MIB (get/set) for IPv4 Only

This defines a portion of the Management Information Base (MIB) for use with network management protocols in the Internet community, which describes managed objects used for managing the Border Gateway Protocol Version 4.

## BGP Peer Group

This is a group of BGP neighbors that share update policies. Instead of defining the same policies for several individual peers, a peer group is defined, and policies are assigned to the group. In this way, the operator is spared repetitive configuration, and the BGP router is saved from parsing the policies sequentially for each neighbor. Instead, the update is formulated once, then the router floods the same update to all neighbors in the group.

## Soft Reconfiguration

Whenever changes are made to the attributes or policies of a route, the BGP TCP session with its neighbor must be reset, for the modified routing behavior to take effect. Resetting the session is not very beneficial: When a session is reset, the routing cache is invalidated, routes disappear, and route instability increases and is propagated throughout the Internet. Resetting the session whenever changes are made is not very efficient and can lead to damage. To prevent this, a mechanism, called soft configuration, is used. It allows attribute reconfigurations to be made without terminating an already established TCP session. Therefore, the routing flap is not cleared, and the impact on the route is minimal.

## Update Source Configuration

An iBGP connection can occur whenever there is a TCP/IP path between the routers. If multiple paths exist between iBGP routers, using a loopback interface as the neighbor's address can add stability to the network. Using loopback interfaces with eBGP routers is not necessary, because eBGP neighbors are usually directly connected.

## AS Path Access-list

This feature filters routes based on AS_PATH information. It is an efficient alternative to listing hundreds of routes one by one, a process that might be required when filtering on a prefix basis.

## Community List

An IP community list that lists routes belonging to a certain community or communities using the specified configuration. Valid list numbers are 1 to 199.

# CHAPTER 3 BGP Module Internal Architecture

## BGP-4+ Protocol Module

The ZebOS-XP BGP-4+ module fully implements the BGP version 4 and version 4+ protocols. BGP-4 supports IPv4, while BGP-4+ supports the IPv6 extensions. The BGP modules are built on NSM and are fully IETF compliant. The BGP-4+ module supports an extensive set of features, including Route Flap Dampening, SNMP support, VPN extensions, Autonomous System Confederations, and Route Reflection. BGP can operate with the VRRP integrated within NSM.

## RFC 4271 Support

BGP supports RFC 4271. The following describes the newly implemented functionality.

### Missing MED

If the Missing Multi Exit Disc (MED) attribute is missing in a received UPDATE message, ZebOS-XP takes the default value of the MED while updating the database.

The `(no) bgp bestpath med remove-recv-med` and `(no) bgp bestpath med remove-send-med` commands remove or send the MED in the UPDATE message to other peers.

The `(no) bgp bestpath med remove-recv-med` command sets the BGP_CFLAG_REMOVE_RCVD flag.

On receiving the UPDATE message, ZebOS-XP checks whether or not this flag is set.

If the flag is set, the MED is not considered for path selection and is not sent in any UPDATE message to other peers.

If the flag is not set, ZebOS-XP considers the MED during route selection and sends MED in the UPDATE message to other peers.

By default, the flag is not set, that is, ZebOS-XP considers the MED during route selection and sends the MED in UPDATE messages to other peers.

The BGP_CFLAG_REMOVE_RCVD flag has higher preference than Missing MED or CONFED. If this flag is not set, only then are the other flags considered.

Setting the flag does not trigger any notifications or UPDATE message with withdraw Network Layer Reachability Information (NLRI) for already-advertised routes and does not cause BGP to re-do route selection for already-selected routes.

Only routes received after setting this flag will be affected by this flag.

For Example:

BGP1 ---------- BGP2 ---------- BGP3

|

BGP4

BGP1 sends an UPDATE message with MED 20 to BGP2; BGP3 sends an UPDATE message with MED 25 to BGP2. In BGP2, the `neighbor A.B.C.D as-set summary-only` command is executed.

Case 1: In BGP2, the `bgp bestpath med remove-recv-med` command is also executed: BGP2 does not consider the MED for route selection. The received routes do not get aggregated here. Also BGP2, does not send the MED in the UPDATE message sent to BGP4.

Case 2: In BGP2, the `no bgp bestpath med remove-recv-med` command is executed: BGP2 considers the MED for route selection. The received routes get aggregated here. Also, BGP2 sends the MED in the UPDATE message sent to BGP4.

The `(no) bgp bestpath med remove-send-med` command sets the BGP_CFLAG_REMOVE_SEND flag.

On sending UPDATE message, ZebOS-XP checks whether or not this flag is set.

This flag reflects in the forwarding process. If the flag is set, the MED is considered for path selection, but the MED is not sent in any UPDATE message to other peers.

If the flag is not set, ZebOS-XP considers the MED during route selection, and while forwarding, sends the MED in the UPDATE message to other peers.

The BGP_CFLAG_REMOVE_SEND flag has higher preference than Missing MED or CONFED. If this flag is not set, only then are the other flags considered.

Setting the flag does not trigger any notifications or UPDATE message with withdraw NLRI for already-advertised routes. Also, the flag setting does not cause BGP to re-do route selection for already-selected routes.

Only routes sent after setting this flag will be affected by this flag.

For Example:

BGP1 ---------- BGP2 ---------- BGP3

|

BGP4

BGP1 sends an UPDATE message with MED 20 to BGP2; BGP3 sends an UPDATE message with MED 25 to BGP2. In BGP2, the `neighbor A.B.C.D as-set summary-only` command is executed.

Case 1: In BGP2, the `bgp bestpath med remove-send-med` command is executed: BGP2 considers MEDs for route selection and aggregates the routes. However, BGP2 does not send the MED in the UPDATE message sent to BGP4.

Case 2: In BGP2, the `no bgp bestpath med remove-send-med` command is executed. BGP2 also considers the MED for route selection and aggregates the routes. Also, BGP2 sends the MED in the UPDATE message sent to BGP4.

Note: If either of the above-described commands are executed after a few routes are exchanged, the flag setting or unsetting will affect only the routes learnt afterward: the already-learnt routes will not be affected by dynamic execution of these commands.

## Hold Timer Configuration on Per-Connection Basis

The Connect Retry Timer is used by a BGP speaker to establish a TCP connection with its peer. The `neighbor connection-retry-time` command is used to set the connection retry time for a specific BGP neighbor.

## BGP Speaker Configuration to Prepend AS Number Multiple Times in AS-PATH

Per RFC 4271, Section 5.1.2, BGP can be configured to prepend its AS number multiple times in the AS_PATH. Normally, when one BGP speaker sends an UPDATE message to another BGP speaker on eBGP, it prepends its ASN once in the AS_PATH::AS_SEGMENT before sending it. But, based on configuration, a BGP speaker prepends its ASN to its peer as many times as it is configured. This is only valid for eBGP sessions.

## Atomic Aggregate Processing

ZebOS-XP supports atomic aggregate processing behavior per RFC 4271.

## Next-Hop Calculation

Next-hop calculation is supported per RFC 4271, Section 5.1.3. The following is an example of this calculation using the topology below.

```
  iBGP              |       eBGP
BGP1 ---------   BGP2 ------------ BGP3
                    |
```

Case 1: If BGP2 learns a route from BGP1 (iBGP route), or the route to be announced is locally generated, BGP2 checks whether BGP3 has the same subnet as that of the received next-hop. If BGP3 has the same subnet, BGP2 sends the update unchanged to BGP3. The constraint is that BGP3 should have advertised all its connected routes to BGP2 earlier. BGP2 uses BGP1's interface address as the next hop while sending an update to BGP3.

Case 2: Consider the above topology, but there is an eBGP session between BGP1 and BGP2. If BGP2 receives an update from BGP1 (which is an external peer), BGP2 checks whether BGP3 has the same subnet as that of the received next-hop: if so, BGP2 sends an update with an unchanged next-hop to BGP3. The constraint is that BGP3 should have advertised all its connected routes to BGP2 earlier. BGP2 uses BGP1's interface address as next-hop while sending an update to BGP3.

Case 3: BGP2 sends one of its interface's address as the next hop in an UPDATE message to BGP3, provided that BGP3 also has a similar subnet on one of its interfaces. Because no two interfaces can be configured with same subnet in one router, the outgoing interface is always used as the next hop.

## Cease Notification

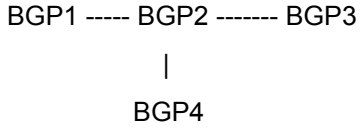
A BGP peer can close its BGP connection by sending a Cease Notification message, as long as there are no fatal errors, as specified in RFC 4271. Upon receiving notifications with the following subcodes, the notification is logged, and the session is reset: BGP_NOTIFY_CEASE_CONN_COLLISION_RES (7) and BGP_NOTIFY_CEASE_OUT_OF_RESOURCES (8).

## FSM Error Handling

BGP Finite State Machine (FSM) error handling is in compliance with RFC 4271.

## Route Selection

ZebOS-XP route selection of received routes is as per RFC 4271, Section 9.1.2. The following is an example of this feature using the topology below.




Case 1: A route is sent from BGP1 (for example, 10.10.10.0 with next-hop, 5.6.7.8) to BGP2. If, in BGP2, there is no route to reach the received next-hop, 5.6.7.8, the received route is not selected by BGP1 and is not forwarded to BGP3 or BGP4.

Case 2: A route is sent from BGP1 (for example, 10.10.10.0 with next-hop, 5.6.7.8) to BGP2. A static route is created for the prefix, 5.6.7.0, using one of its interfaces as the next hop. BGP2 selects the received route and forwards it to the other BGP speakers, BGP3 and BGP4.

Case 3: A route is sent from BGP1 (10.10.0.0 with next-hop, 5.6.7.8) to BGP2. A route is sent from BGP3 (5.6.7.0 with next-hop, 10.10.10.10) to BGP2. BGP2 does not select either of the routes because the two routes are mutually recursive routes. (Section 9.1.2.1.)

## Tie-Breaking Procedure

The tie-breaking procedure behavior is per RFC 4271, Section 9.1.2.2. The tie-breaking algorithm considers all equally preferable routes to the same destination, then selects routes to be removed from consideration. The algorithm terminates when only one route remains in consideration.

## Overlapping Routes

A BGP speaker may transmit routes with overlapping NLRI to another BGP speaker. NLRI overlap occurs when a set of destinations are identified in non-matching multiple routes.

ZebOS-XP BGP handles overlapping routes per RFC 4271, Section 9.1.4. The following is an example of this feature using the topology below.


Case 1: From BGP1, a route (for example, 10.10.10.0) is sent to BGP2; from BGP3, a route (for example, 10.10.0.0) is sent to BGP2. With no aggregation-related configuration, BGP2 installs both routes and sends these routes to BGP4.

Case 2: From BGP1, a route (for example, 10.10.10.0) is sent to BGP2; while from BGP3, another route (for example, 10.10.0.0) is sent to BGP2. The aggregation in BGP2 is configured: BGP2 installs the less specific route (for example, 10.10.0.0) and sends this route to BGP4.

Case 3: From BGP1, a route (for example, 10.10.0.0) is sent with the ATOMIC_AGGREGATE attribute in the UPDATE message to BGP2. From BGP3, a route (for example, 10.10.10.0) is sent to BGP2. BGP2 does not alter the ATOMIC_AGGREGATE attribute and sends the attribute to BGP4. BGP2 does not forward the more specific route to BGP4.

## Aggregating Route Information

ZebOS-XP BGP handles aggregating route information per RFC 4271, Section 9.2.2.2.

Routes with the same MED are aggregated, even if the next hops are different. BGP checks for the same next-hop while aggregating the routes. While aggregating, checking for the same MEDs is done only when the BGP_CFLAG_REMOVE_MED flag is not set. The following is an example of this feature using the topology below.


Case 1: From BGP1, a route (for example, 10.10.10.0 with MED 10) is sent to BGP2; from BGP3 a route (for example, 10.10.0.0 with MED 20) is sent to BGP2. BGP2 is configured to aggregate the routes to 10.0.0.0. BGP2 does not aggregate the routes (because the MEDs are different), installs both routes, and sends both routes to BGP4.

Case 2: From BGP1, a route (for example, 10.10.10.0 with MED 10) is sent to BGP2; from BGP3, a route (for example, 10.10.0.0 with MED 10) is sent to BGP2. BGP2 is configured to aggregate the routes to 10.0.0.0. BGP2 installs the aggregated route (10.0.0.0) and send this route to BGP4.

Case 3: The above test cases also apply to different attributes, such as NEXT-HOP, ORIGIN, AS_PATH.

## BGP MD5 Verification

To enhance BGP security, an MD5 digest is carried in a TCP segment, which acts as a signature for the segment, to incorporate information known only to the connection end-points.

# RFC 4760 Support

The IPV6-SI-MPLS_LABEL and IPV6-SI_BGP_MPLS_VPN I-SI combinations are supported per RFC 4760. These Subsequent Address Family Identifier (SI) combinations accommodate 6PE and 6VPE. The following are examples of ZebOS-XP BGP error handling and capability advertisement.

## Error Handling

The following is an example of error handling compliant with RFC 4760, Section 7, using the topology below.

```
                eBGP/iBGP
BGP1    ------------------    BGP2
```

1. Multiple routes are sent from BGP1 to BGP2. An UPDATE message is sent with an incorrect attribute. BGP2 sends a notification with the sub-code, Optional Attribute Error, and deletes all routes received from BGP1.
2. A route is sent from BGP1 to BGP2 with a reserved SI value in the UPDATE message. BGP2 sends a notification with the sub-code, Unsupported Optional parameter.
3. Step 2 is repeated with MP_REACH_NLRI and MP_UNREACH_NLRI.
4. A Route-Refresh message is sent from BGP1 with a reserved SI to BGP2. BGP2 ignores the received Route-Refresh and does not send any routes to BGP2.

## Capability Advertisement

The following is an example of capability advertisement compliant with RFC 4760, Section 8, using the topology below.

```
BGP1    ------------------ BGP2
```

Case 1: An OPEN message, with a Mutli-protocol Capability message containing correct I and SI values, is sent from BGP1. BGP2 is configured so that it also exchanges similar Multi-Protocol capability with BGP1. The session is successfully established.

Case 2: An OPEN message, with a Mutli-protocol Capability message containing correct I and SI values, is sent from BGP1. No configuration is made on BGP2 to exchange its Multi-Protocol capability with BGP1. BGP1 sends a notification with “Unsupported Capability”. BGP1 sends an OPEN message without MP capabilities, and the session is successfully established.

Case 3: An OPEN message, with a Mutli-protocol Capability message containing correct I and UNICAST_MULTICAST SI values, is sent from BGP1. BGP2 sends a notification with “Unsupported Capability”, and the session is not established.

Case 4: An OPEN message, with a Mutli-protocol Capability message containing correct I and reserved SI values, is sent from BGP1. BGP2 sends a notification with “Unsupported Capability”, and the session is not established.

# RFC 4273 Support

BGP MIB, data structures, and functions are in accordance with RFC 4273 to define BGP notifications, instead of BGP traps.

# CHAPTER 4 Route Refresh

## Introduction

Route-Refresh Capability in BGP allows the dynamic exchange of route-refresh requests between BGP speakers and subsequent re-advertisement of the respective Adj-RIB-Out. To advertise the Route-Refresh Capability to a peer, a BGP speaker uses BGP Capabilities Advertisement [BGP-CAP]. By advertising the Route-Refresh Capability to a peer, a BGP speaker conveys to the peer that the speaker is capable of receiving, and properly handling, the ROUTE-REFRESH message from the peer.

ZebOS-XP BGP supports the Route-Refresh Capability for both the IPv4 and IPv6 address families.

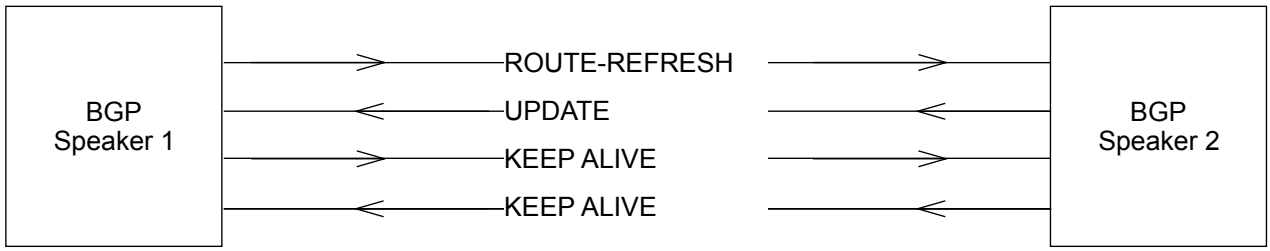
## ZebOS-XP Behavior

This section provides examples of ZebOS-XP behavior when commands related to the Route-Refresh Capability are executed.

### Example 1

The following commands generate soft reset and send a ROUTE-REFRESH message, and in turn, an UPDATE message is received; also indicated in the illustration that follows.

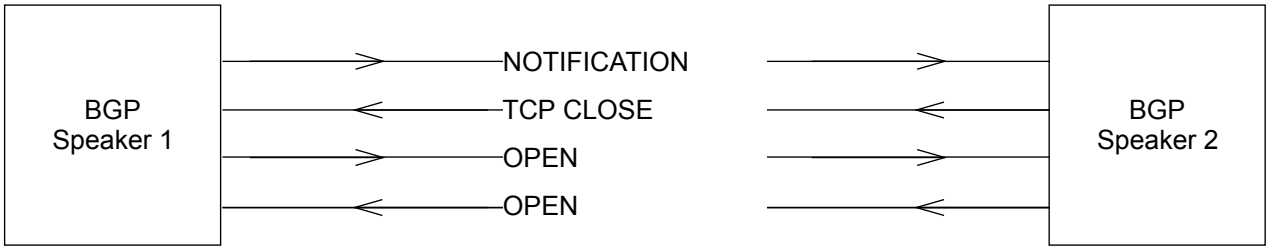
| IPv4 Commands | IPv6 Commands |
|---|---|
| clear ip bgp A.B.C.D in | clear bgp ipv6 X:X::X:X in |
| clear ip bgp A.B.C.D soft | clear bgp ipv6 X:X::X:X soft |
| clear ip bgp A.B.C.D soft in | clear bgp ipv6 X:X::X:X soft in |
| clear bgp A.B.C.D in | clear bgp X:X::X:X in |
| clear bgp A.B.C.D soft | clear bgp X:X::X:X soft |
| clear bgp A.B.C.D soft in | clear bgp X:X::X:X soft in |




### Example 2

The following commands cause the BGP speaker to send a NOTIFICATION message to the other speaker, after which the connection is re-established.

| IPv4 Commands | IPv6 Commands |
|---|---|
| clear ip bgp A.B.C.D | clear ip bgp X:X::X:X |
| clear bgp A.B.C.D | clear bgp X:X::X:X |




# Command Functionality

This section describes the functionality of commands related to the Route-Refresh Capability.

### (no) neighbor A.B.C.D | X:X::X:X | TAG capability route-refresh

This command enables or disables Route-Refresh Capability for an IPv4 or IPv6 neighbor, or all peers in a specified group.

### clear ip bgp A.B.C.D | X:X::X:X in, clear ip bgp A.B.C.D | X:X::X:X soft in

These commands clear the adjacent RIB-IN (Adj-RIB-IN) and trigger a ROUTE-REFRESH message to the next BGP speaker. The speaker that receives this ROUTE-REFRESH message sends the UPDATE message, so that the speaker that sent the ROUTE-REFRESH message updates the Adj-RIB-IN.

### clear ip bgp A.B.C.D | X:X::X:X soft

This command clears the Adj-RIB-IN and Adj-RIB-OUT, and triggers a ROUTE-REFRESH message to the next BGP speaker. The speaker that receives this ROUTE-REFRESH message sends the UPDATE message, so that the speaker that sent the ROUTE-REFRESH message updates the Adj-RIB-IN.

### clear ip bgp A.B.C.D | X:X::X:X out, clear ip bgp A.B.C.D | X:X::X:X soft out

These commands clear outgoing advertised routes.

# CHAPTER 5 BGP Extended Community Attribute

The BGP Extended Community Attribute provides a mechanism for labeling information carried in BGP-4, and provides an extended range of routes and the addition of a Type field. This allows policy usage based on the application for which the community value is used. It also provides a means to specify whether a particular community is transitive, or non-transitive, across an Autonomous System (AS) boundary.

Also supported are:

- Opaque extended community type
- Four-byte AS specific extended community attribute to support BGP speakers that use four-byte AS numbers

## Functionality

This section describes how ZebOS-XP handles the BGP Extended Community Attribute.

When BGP receives a non-transitive extended community attribute, it does not pass it to eBGP neighbors. ZebOS-XP logs all non-transitive attributes.

When BGP receives a transitive extended community attribute, it passes it to the neighboring BGP peers.

ZebOS-XP supports 4-byte ASs in the route distinguisher (RD) configuration with Type 0x02, as specified in RFC 4364.

When 4-byte AS capability is enabled, the Virtual Routing and Forwarding (VRF) configuration is created with Type 2 for extended community attributes.

When an operator attempts to change the AS capability from 2 to 4, or 4 to 2, a prompt instructs the operator to remove the VRF configuration (if it exists), and the operator must reconfigure, as required. The reason for this behavior is the RD configuration would have been created with the current (2-byte or 4-byte) capability, and must be reconfigured before attempting to change the capability.

VPN route exchanges between 4-byte AS and 2-byte AS BGP occur only when Type 1 route target configuration is done. When the type is 0 or 2, the routes are denied because of MPLS VPN inbound filtering.

# BGP Extended Community Attributes and Sub-Types

This section describes BGP Extended Community Attributes and their sub-types.

## BGP Extended Community Types

The Extended Communities Attribute is a transitive optional BGP attribute with Type Code 16. Each extended community is encoded as an 8-byte quantity, with a type field of 1 or 2 bytes (high-order bytes) and a value field of 6 bytes. When the type field is only 1 byte, it is the regular type, if 2 bytes, it is the extended type.

The value of the low-order byte of the extended type (of the type field) indicates the sub-types (Route Target Community and Route Origin Community).

Route Target Community is for specifying one or more routers that can receive a set of routes advertised by BGP that carry the Extended Community Attribute. Route Origin Community is for specifying one or more routers that inject a set of routes that carry the extended community attribute into BGP.

Following are the standard extended communities defined in RFC4360 and their assigned values.

### Two-Byte AS Specific Extended Community

| Name | Type Value |
|---|---|
| Two-byte AS specific Route Target (Transitive) | 0x0002 |
| Two-byte AS specific Route Origin (Transitive) | 0x0003 |

The value field (of 6 bytes) for AS specific communities has the following format:

AA:NNNN

Where AA is the global administrator sub-field containing the AS number (of 2 bytes), and NNNN (of 4 bytes) is the community within the AS.

### IPv4 Address Specific Extended Community

| Name | Type Value |
|---|---|
| IPv4 address specific Route Target (Transitive) | 0x0102 |
| IPv4 address specific Route Origin (Transitive) | 0x0103 |

The value field (of 6 bytes) for this community has the following format:

x.x.x.x:NN

Where x.x.x.x is the IPv4 address (4 bytes), and NN is the community number (2 bytes).

## Opaque Extended Community

The value of the high-order byte of this extended type is either 0x03 or 0x43. The low-order byte of this extended type is used to indicate sub-types. The type values for the transitive and non-transitive communities of this class are 0x0300-0x03ff and 0x4300-0x43ff, respectively. The value of the sub-type field that defines the value field is assigned by IANA.

Depending on the attribute types (transitive or non-transitive), the attributes are sent with routing updates when redistribution occurs. If the type is opaque and transitive, the attribute transparently passes to the other AS; if non-transitive, the attribute is dropped.

## Four-Byte AS Specific Extended Community

Four-byte AS specific extended community is similar to two-byte AS specific extended community, except it can carry a four-byte AS number (draft-rekhter-as4octet-ext-community-02.txt defines new community attributes for BGP speakers with a non-mappable 4-byte AS number).

The non-transitive communities of this class have the range, 0x4200-0x42ff. The value of the sub-type field for this class is not defined.

The transitive communities of this class have the range, 0x0200-0x42ff. BGP supports the following two sub-types for AS specific and IP specific in transitive mode: route target and route origin. The assigned type values are given below.

| Name | Type Value |
|---|---|
| Four-byte AS specific Route Target (Transitive) | 0x0202 |
| Four-byte AS specific Route Origin (Transitive) | 0x0203 |

The value field (of 6 bytes) for this community has the following format:

AAAA:NN

Where AAAA is the global administrator sub-field containing the AS number (4 bytes), and NN is the community within the AS (2 bytes).

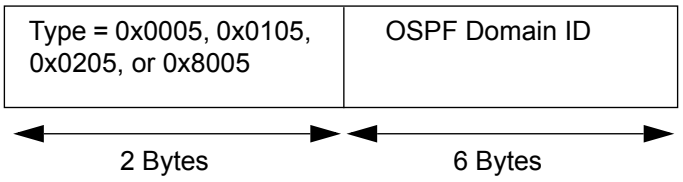
## Non-Transitive Extended Community

BGP can accept the non-transitive extended-community attributes, and suppresses these attributes, while passing across ASs.

The non-transitive communities for two-byte AS and IPv4 specific extended-community classes are defined in the 0x4000-0x40ff and 0x4100-0x41ff ranges, respectively.

## OSPF Domain ID

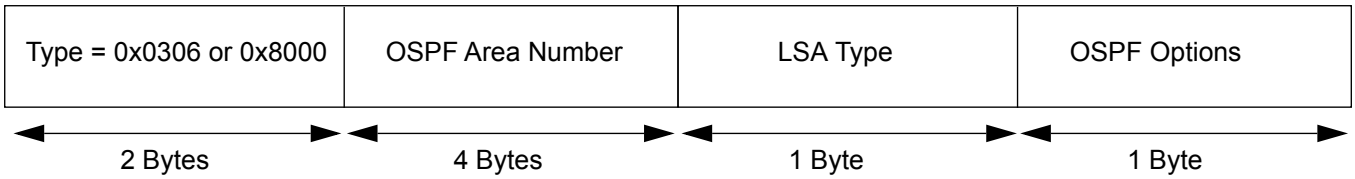
The OSPF domain ID identifies the domain of a specific OSPF prefix in the MPLS VPN backbone. If the domain ID of the route does not match the domain ID on the receiving PE, the route is translated to the external OSPF route (LSA Type-5) with metric-type E2, assuming the route was received in the VRF table. All routing between OSPF domains is via Type-5 LSAs.




- Type 0x0005: AS type domain ID extended community
- Type 0x0105: IP type domain ID extended community
- Type 0x0205: AS4 type domain ID extended community
- Type 0x8005: used for backward compatibility; treated as 0x0005

## OSPF Route Type

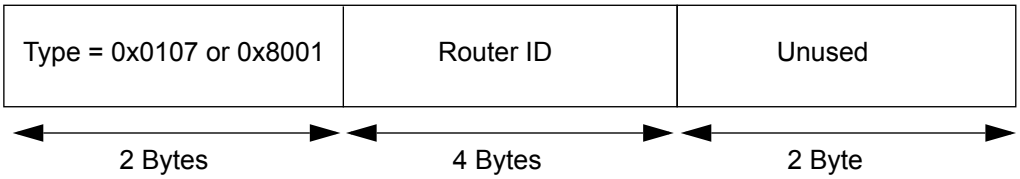
The OSPF route type propagates OSPF route type information across the MP-iBGP backbone.




- Type 0x0306: Opaque type extended-community attribute for OSPF route type
- Type 0x8000: used for backward compatibility; treated as 0x0306

## OSPF Router ID

The router ID is given in Router OSPF mode. This information is sent to BGP via NSM. BGP sends this attribute in the extended-community attribute to the remote PE.




- Type 0x0107: IP type extended community for OSPF Router ID
- Type 0x8001: used for backward compatibility; treated as 0x0107

# CHAPTER 6 Four-Byte ASN

## Overview

ZebOS-XP BGP encodes and decodes the autonomous system number (ASN) in four-byte format, by default. However, exchanging four-byte capability with its peer can be controlled through configuration.

This chapter provides various BGP system configurations, including interactions between BGP systems that support two-byte and four-byte capability. BGP speakers that support four-byte ASNs are referred to as four-byte BGP; BGP speakers that support two-byte ASNs are referred to as two-byte BGP.

## Interaction Between Four-Byte BGP Systems

The following describes interaction between two four-byte BGP systems, BGP1 and BGP2.

### Forming BGP Session

- If both four-byte BGPs are configured to advertise their four-byte ASN capability (using global configuration), they exchange the four-byte ASN capability message in the OPEN message.
- If the configured ASN is in two-byte format, it is sent in the My ASN field of the OPEN message with a new ASN capability message.
- If the configured ASN is in non-mappable four-byte format, the ASN is sent in the ASN capability message of the OPEN message, and the My ASN field of the OPEN message has AS_TRANS.
- If BGP1 is configured with a non-mappable four-byte ASN, and BGP2 is configured with a mappable ASN, by default, AS4_PATH capability is turned off, BGP2 sends its ASN in the My ASN field of the OPEN message, and no new capability messages are sent to BGP1. The adjacency is formed between the two BGPs. (In this case, BGP2 acts as a two-byte BGP).
- If BGP2 sends an OPEN Message with AS_TRANS in the My ASN field, and AS_TRANS as the ASN in the new capability message, BGP1 sends a notification message with the sub-code, Bad Peer AS.

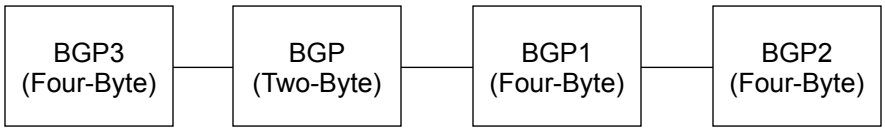
### Update Messages Between Four-Byte BGP Speakers

The following lists information about UPDATE messages between four-byte BGPs, after the adjacency is formed.

- The four-byte BGPs send UPDATE messages to each other with their ASN prepended to the AS_PATH attribute, irrespective of whether their configured ASN is in four- or two-byte format.
- While a four-byte BGP sends an UPDATE to another four-byte BGP peer, it always does a conversion of two-byte values of AS-numbers into four-byte and sends the converted values in the AS_PATH attribute. This is the case when a four-byte BGP speaker receives updates from a two-byte BGP speaker in one end, and advertises the routes to a four-byte BGP peer.
- Between two four-byte BGPs, AS_PATH/AGGREGATOR only carry ASNs in 4-byte format.

## Example

As shown in the following topology, if (four-byte) BGP1, with a four-byte unmappable ASN receives an UPDATE message with AS4_PATH/AS4_AGGREGATOR from a two-byte (BGP), BGP1 does the actions listed below




- Converts all ASNs of AS_PATH/AGGREGATOR to four-byte ASNs.
- Removes the AS_TRANS values (if any) from AS_PATH/AGGREGATOR attributes.
- Appends all four-byte ASNs of AS4_PATH to AS_PATH.
- Prepends its ASN in AS_PATH and sends an UPDATE message to BGP2.
- If the number of AS_TRANS in AS_PATH is less than the number of ASNs present in AS4_PATH, BGP1 takes the four-byte ASN from AS4_PATH for each AS_TRANS in AS_PATH, and appends all other four-byte ASNs at the end.
- If the number of AS_TRANS in AS_PATH is greater than the number of ASNs present in AS4_PATH, AS4_PATH and the corresponding AS_TRANS of AS_PATH is ignored.
- The four-byte ASN is taken from AS4_AGGREGATOR on encountering AS_TRANS in the AGGREGATOR attribute.

BGP1 follows the above algorithm while sending an UPDATE to BGP2, even if it is configured with a two-byte ASN.

Note: 1) A four-byte BGP never sends AS4_PATH/AS4_AGGREGATOR to other four-byte BGP peers, and
2) If BGP1 is configured with a mappable ASN, and `bgp extended_asn_cap` is not performed, BGP1 acts as a two-byte BGP implementation.

# Interaction Between Four-Byte and Two-Byte BGP Systems

The following describes interaction between a four-byte BGP system and a two-byte BGP system.

## Forming BGP Session

- The four-byte BGP speaker, if configured with an unmappable four-byte ASN, sends an OPEN message to the two-byte BGP speaker with AS_TRANS in the My ASN field of the OPEN message. The four-byte BGP sends a four-byte ASN new capability message in the OPEN message with the ASN value in the capability message.
- The two-byte BGP speaker takes only the AS_TRANS as its peer's ASN and forms an adjacency.
- The four-byte BGP, if configured with only a two-byte ASN, sends an OPEN message with the configured two-byte ASN in the My ASN field of the OPEN message. As such, the new capability four-byte ASN is not sent to the OPEN message.
- If the four-byte BGP receives an OPEN message from the two-byte BGP, with a value of AS_TRANS (23456) in the My ASN field, it sends a notification message with the sub-code, Bad Peer AS.

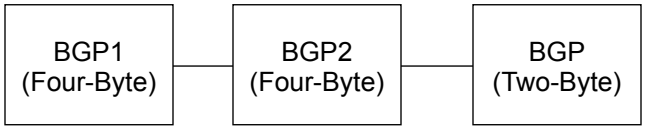
## Update Messages Between Four-Byte and Two-Byte BGP Speakers

The following lists information about UPDATE messages between four-byte and two-byte BGPs, after the adjacency is formed.

- If a four-byte BGP is configured with a two-byte ASN, it sends an UPDATE message with its ASN prepended in the AS_PATH/AGGREGATOR attribute.
- If a four-byte BGP is configured with an unmappable four-byte ASN, it sends the UPDATE message with its ASN in AS4_PATH/AS4_AGGREGATOR and AS_TRANS in AS_PATH/AGGREGATOR.

## Example

As shown in the following topology, when (four-byte) BGP2, configured with a four-byte unmappable ASN, receives an UPDATE message with AS4_PATH/AS4_AGGREGATOR from (four-byte) BGP1, BGP2 does the actions listed below.




- A new attribute, AS4_PATH/AS4_AGGREGATOR, is added.
- The four-byte ASNs of AS_PATH/AGGREGATOR are added to AS4_PATH/AS4_AGGREGATOR, and correspondingly, AS_TRANS is added in AS_PATH/AGGREGATOR.
- It prepends its four-byte ASN in AS4_PATH and four-byte ASN in AS4_AGGREGATOR.

  Note: When BGP2 is configured with a two-byte ASN, the above algorithm is used, but instead of prepending its four-byte ASN in AS4_PATH and four-byte ASN in AS4_AGGREGATOR, it prepends its ASN in AS_PATH and copies its ASN in AGGREGATOR, while sending an UPDATE to the two-byte BGP.

- The two-byte BGP does not process the AS4_PATH/AS4_AGGREGATOR and considers only AS_PATH/ AGGREGATOR attributes. However, the two-byte BGP passes the AS4_PATH/AS4_AGGREGATOR attributes as is to its peer, while sending an UPDATE message (because AS4_PATH/AS4_AGGREGATOR are optional transitive attributes) with the PARTIAL bit set.
- If BGP2 receives an UPDATE message with the AS_CONFED/ AS_CONFED_SEQ attribute, along with AS4_PATH/AS4_AGGREGATOR, it does not process it.

# AS PATH and AGGREGATOR Reconstruction

The following describes the reconstruction of the AS PATH and AGGREGATOR attributes.

A two-byte BGP receiver is not affected by AS PATH reconstruction because it ignores the two new attributes, AS4_PATH/AS4_AGGREGATOR.

A four-byte BGP speaker, on receiving an UPDATE message with no AS4_PATH/AS4_AGGREGATOR, reconstructs the AS PATH into four bytes, only when it peers with another four-byte BGP peer that exchanges four-byte capability messages.

A four-byte BGP, on receiving an UPDATE with AS4_PATH/AS4_AGGREGATOR, functions as follows:

- If AS_TRANS is not present in the AGGREGATOR attribute, a four-byte BGP ignores the AS4_AGGREGATOR and AS4_PATH attributes, and uses the AGGREGATOR attribute only for aggregation information. In this case, only the AS_PATH attribute is considered for AS path information.

- If the AGGREGATOR attribute contains AS_TRANS, AS4_AGGREGATOR is taken as information for aggregating the route. For path construction, both the AS_PATH and AS4_PATH attributes are considered, according to the following rules:
  - If the number of ASNs in the AS_PATH attribute is less than the number of ASNs in the AS4_PATH attribute, a four-byte BGP ignores AS4_PATH information.

    Example:

    ```
    AS_PATH:   2 23456
    AS4_PATH: 70000 70001 70002
    ```

    Only AS_PATH is taken for path reconstruction.
  - If the number of ASNs in the AS_PATH attribute is greater than number of ASNs in the AS4_PATH attribute, there are multiple AS_TRANS numbers in AS_PATH. Each ASN is taken from AS_PATH. The AS_TRANS numbers go to AS4_PATH and take the first four-byte ASN, and likewise, take all 4-octet ASNs. The remaining four-byte ASNs are appended to AS_PATH. Thus, reconstruction is completed.

    Note: A valid AS_CONFED_SEQUENCE or AS_CONFED_SET path segment is prepended if it is either the leading path segment or adjacent to a prepended path segment.

    Examples:

    ```
    AS_PATH: 2,3,6, 8, 23456, 10, 23, 23456
    AS4_PATH: 65356, 77777
    ```
  - If the number of ASNs in the AS_PATH and AS4_PATH attributes is the same, all of the AS numbers are probably AS4-byte unmappable numbers. The count of AS_TRANS numbers in AS_PATH and count of AS numbers in AS4_PATH are checked. If they are the same, only AS information from AS4_PATH is reconstructed (this case is for AS num == AS4 num). If the number of AS_TRANS in AS_PATH is less than the number in the AS4_PATH values, an erroneous action was done by previous speakers. However, the non-AS_TRANS AS values are taken from AS_PATH, then prepended with the AS4 values, and the AS path count is recomputed.

    Examples:

    ```
    AS_PATH:   23 456, 23456, 23456
    AS4_PATH: 777777, 66666, 88888
    ```

    Or

    ```
    AS_PATH: 2, 3, 4, 23456
    AS4_PATH: 88888, 99999, 70000, 80000
    ```

# CHAPTER 7 BGP Confederation

Nodes running iBGP protocols must be interconnected forming a full mesh among each other. Confederation solves the iBGP full mesh network complexity and inefficiency by splitting a large Autonomous System (AS) domain into smaller AS domains, called member-ASs. Member-ASs can form eBGP connections among themselves, thus preventing full-mesh connections among each iBGP running node. All eBGP sessions within a single confederation of AS-members are called intra-confederation eBGP sessions. These sessions behave like iBGP sessions, in terms of preserving NEXT_HOP and MED, but modify the AS_PATH with AS_CONFED_SEQUENCE or AS_CONFED_SET. When updates are sent from the confederation boundary router to an external AS domain, the AS_CONFED_* segments are removed from the AS_PATH, and the confederation ID is prepended as the AS number. Thus, eBGP speakers outside a confederation are unaware of confederation ASs.

Note: Within a member-AS, a full iBGP mesh or a route-reflector configuration is required.

The commands `bgp confederation identifier`, `bgp confederation peers`, and `bgp bestpath compare-confed-aspath` are used to support BGP Confederation

## ZebOS-XP Behavior

This section describes how ZebOS-XP handles BGP Confederation.

### MED and LOCAL-PREFIX

The following describes MED and LOCAL-PREFIX handling for best-path selection and local-AS considerations.

In the MED check section of the ZebOS-XP best-path selection code, it checks if confed_as is present by checking AS_PATH_LEN is greater than zero, even when AS_PATH_COUNT is zero: This check is performed if MED-COMPARE-CONFED is set. Thus, ZebOS-XP checks for MED values when they are from the same confederation sequence or from the same AS-member of a confederation.

The ZebOS-XP aspath_cmp_left_confed() function ensures the correct confederation AS-member value is taken during the MED comparison.

### Aggregation

ZebOS-XP performs AS_PATH related checks for confederation for appending and aggregating AS_CONFED_* related segments and adds the confederation identifier as the AS value in AS_PATH segment at the confederation boundary.

### Intra-Confederation eBGP and iBGP Peering

In ZebOS-XP, bgp_info_cmp() ensures that intra-confederation eBGP peering is considered the same as iBGP peering, in terms of handling LOCAL-PREFIX, MED, and NEXT_HOP.

# AS-PATH Checking

1. ZebOS-XP checks that AS_CONFED_SEQUENCE is set as the first segment when a member-AS generates routes within the confederation if the AS_PATH is not empty (AS_PATH_LEN is non-zero).
2. The originating speaker includes an empty AS_PATH attribute in all UPDATE messages sent to BGP speakers residing within the same member-AS. (An empty AS_PATH attribute's length field contains the value, zero).
3. Subsequently, another member-AS adds its AS as the left-most element in the CONFED segment.
4. When updates travel from a confederation AS to an external AS that is not part of any confederation, or part of a different confederation, the sender removes the CONFED_SEQUENCE, and puts the confederation identifier as the left-most value in AS_PATH. If the AS_PATH is empty, the speaker creates an AS_SEQUENCE segment and add its confederation identifier as the first element.

## Intra-Confederation

If the first path segment of the AS_PATH is Type AS_CONFED_SEQUENCE, the local system prepends its own member-AS number as the last element of the sequence (left-most position with respect to the position of bytes in the protocol message). If prepending causes an overflow in the AS_PATH segment (that is, more than 255 ASs), the aspath_prepend function prepends a new segment of Type AS_CONFED_SEQUENCE and prepends its own AS number to this new segment.

If the first path segment of the AS_PATH is not Type AS_CONFED_SEQUENCE, the local system prepends a new path segment of Type AS_CONFED_SEQUENCE to the AS_PATH, including its own member-AS number in that segment.

If the AS_PATH is empty, the local system creates a path segment of Type AS_CONFED_SEQUENCE, places its own member-AS number into that segment, and places that segment into the AS_PATH.

# Error Handling

A BGP speaker does not transmit updates containing AS_CONFED_SET or AS_CONFED_SEQUENCE attributes to peers that are not members of the local confederation.

A BGP speaker does not process an UPDATE message received with an AS_PATH attribute that contains AS_CONFED_SEQUENCE or AS_CONFED_SET segments from a neighbor not located in the same confederation. If a BGP speaker receives this type of UPDATE message, it treats the message as having a malformed AS_PATH.

A BGP speaker does not process an UPDATE message received from a confederation peer not in the same member-AS that does not have AS_CONFED_SEQUENCE as the first segment. If a BGP speaker receives this type of UPDATE message, it treats the message as having a malformed AS_PATH.

# CHAPTER 8 Next-hop Tracking

In ZebOS-XP BGP, Next-hop Tracking (NHT) notifies BGP asynchronously whenr there is a change in the IGP routes. NHT reduces the convergence time of BGP routes when IGP routes are changed. ZebOS-XP supports the Address Family Indicator/Subsequent Address Family Indicator (I-SI): IPv4-UNICAST and IPv6-UNICAST.

## NHT Capabilities

ZebOS-XP BGP provides the following capabilities:

- Commands that enable and disable the NHT feature.
- BGP accepts next-hop change triggers from NSM, updates the BGP RIB with the changes, and informs peers, if necessary.
- BGP disables the NHT feature if IGP routes are fluctuating frequently.
- The BGP process waits for a configured or default time after receiving a next-hop change trigger from NSM, before it updates the BGP RIB. Commands are provided to set the time delay.

Note: NHT is turned off by default.

## BGP RIB Scanning Operations with NHT

NSM notifies BGP asynchronously about IGP route changes. The NHT feature is not enabled automatically by default. This feature is enabled only when the user executes the `bgp nexthop trigger enable` command, as described later in this chapter.

The following figure shows how BGP RIB scanning functions in ZebOS-XP.

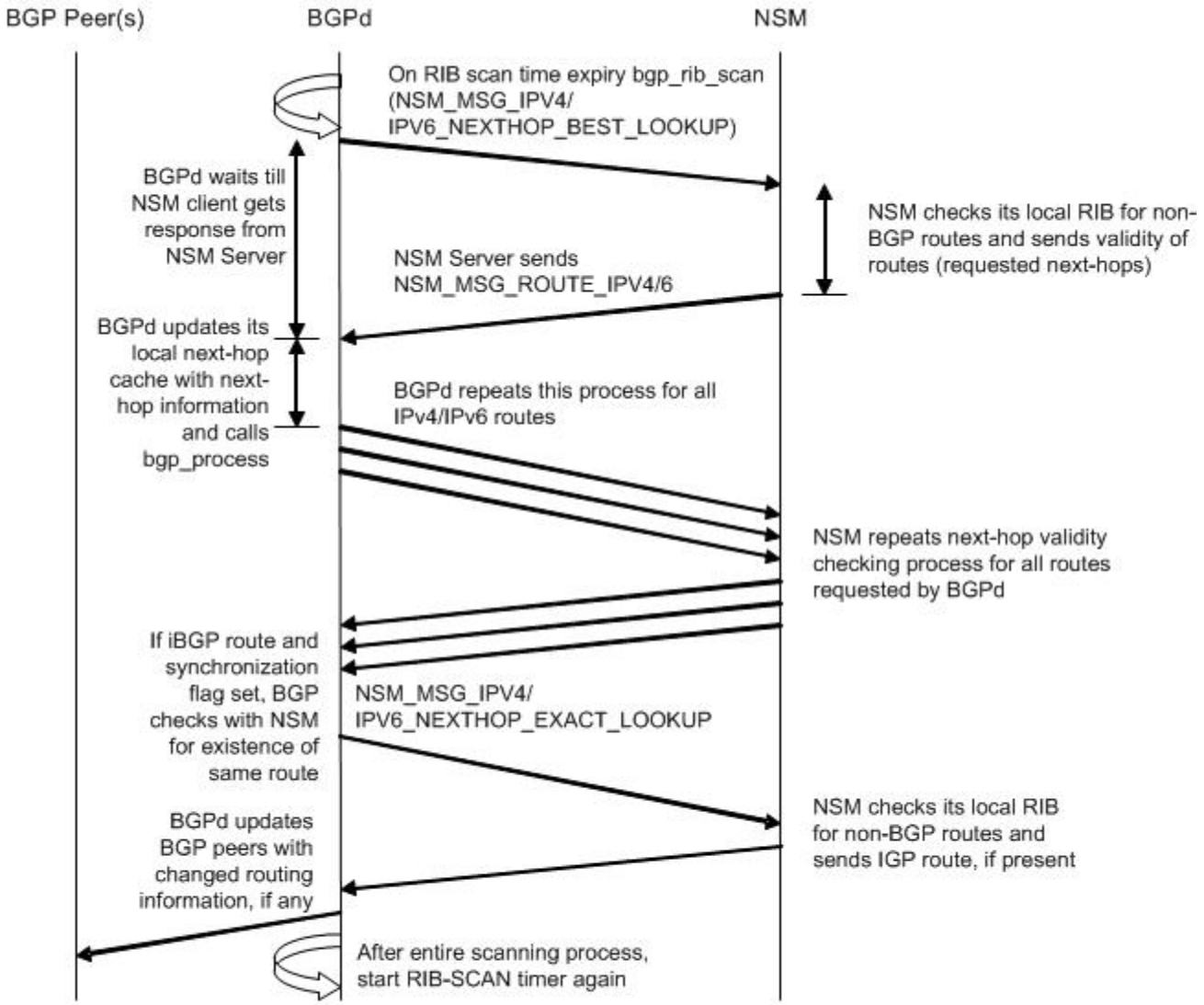



**Figure 8-1: BGP RIB Scanning with Next-hop Tracking**

# BGP NHT Delay Timer Operation

The BGP process supports user configuration of the next-hop delay time, waiting for the configured amount of time after receiving NSM_MSG_ROUTE_IPV4/IPV6 with the NSM_MSG_NEXTHOP_CHANGE message flag before it starts its internal process. The allowable range of next-hop delay times is from 1 to 100 seconds. The default value is 5 seconds.

The BGP process sets the next-hop delay timer for the default time (5 seconds) if next-hop delay time is not configured, or if the delay time is unconfigured.

### Enabling and Disabling NHT

The `bgp nexthop trigger enable` command enables the NHT feature, and the `no bgp nexthop trigger enable` command disables the NHT feature. NHT is not enabled by default. However, when the NHT feature is enabled, and no delay-time interval is configured, the default next-hop delay time-interval of 5 seconds is used.

If the `no bgp nexthop trigger enable` command is given when the NHT function is executing, an error is sent to the user, and the NHT is not disabled. But, if the NHT timer is running, the NHT timer stops, and the NHT feature is disabled.

If NHT is enabled after certain routes are learned, next-hop registration for the next-hops of all BGP selected routes is performed after the NHT feature is enabled using the CLI. Also, when NHT is disabled, the de-registration of next-hops of the already existing BGP routes is done after the NHT feature is disabled using the CLI.

### Setting and Resetting the Delay-time for NHT

Use the following command to set or reset the delay-time for NHT:

```
(no) bgp nexthop trigger delay <1-100>
```

The default value of the next-hop delay time is 5 seconds. Using the `no bgp nexthop trigger delay` command resets the timer value to its default value. This command is in Configure mode and affects the next-hops of all IPv4 and IPv6 routes. For more information, refer to the *Border Gateway Protocol Command Reference*.

# CHAPTER 9 Graceful Restart and Graceful Reset

The graceful restart mechanism helps minimize the negative effects on routing caused by BGP restart. Graceful restart allows a restarting router, and its neighbors, to continue forwarding packets, without disrupting network performance. Because neighboring routers assist in the restart, the restarting router can quickly resume full operation.

Graceful reset is a further refinement of graceful restart to help ensure smooth restarts when a configuration change forces BGP peer reset.

This chapter discusses how the graceful restart and graceful reset capabilities function in ZebOS-XP BGP, and briefly explains how to use the commands associated with these features.

## Graceful Restart and Graceful Reset

Without the graceful restart capability, during BGP restart all BGP peers detected that a session had gone down and come back up. ZebOS-XP invalidated the associated portion of the IP forwarding cache, did a BGP route re-computation, and generated BGP routing updates. The forwarding tables became corrupted and unstable.

ZebOS-XP provides a way for BGP to minimize the negative effects on routing caused by BGP restart. It allows the restarting BGP router to temporarily retain routing information and continue forwarding packets, while BGP restarts. In this way, even while the routing is rebuilding routing and forwarding tables, the router continues to operate across the TCP connection.

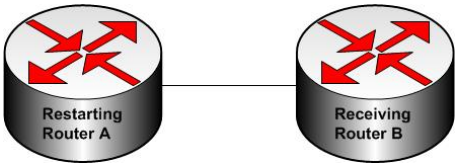
The graceful restart capability extends to the case when a configuration change forces a peer reset.

In addition, graceful restart is available for BGP with MPLS, when BGP is used to distribute MPLS-VPN labels. Without the graceful restart capability, when BGP distributed MPLS-VPN labels, a BGP route withdrawal accompanied the associated label withdrawal. This caused a routing flap and a BGP route re-computation, generating BGP routing updates, and unnecessary disruption in the forwarding tables. Also, when BGP went down, label-switched routers (LSRs) cleared FEC label bindings (for VPN routes) learned from the restarting LSR. As a result, MPLS Forwarding was impacted across the restart.

The graceful restart, graceful reset capability provides a way to save MPLS forwarding states n NSM. It also synchronizes with the VRF table when BGP goes down in the control plane. This feature is supported for the VPNv4 address family.

## How Graceful Restart Works

Using BGP graceful restart, the data-forwarding plane of a router can now continue to process and forward packets, even if the control plane—which is responsible for determining best paths—fails, as shown in the following example.




If (as in the illustration above) two routers (A and B) support the graceful restart capability (capability code 64), the following describes the sequence of events during graceful restart:

1. The BGP process in Router A restarts.
2. Router B detects the TCP session failure with Router A and marks routes from A as stale in its RIB. It also starts the restart timer. The default time is 120 seconds.
3. Router A does not have BGP routes in its RIB at restart and must re-acquire them from its peer routers. However, it retains preserved stale routes in its FIB.
4. Once Router A re-establishes a BGP session with Router B from the open message sent from A—with the restart state bit set, a restart time of n, and forwarding state = set—Router B resets the restart timer and starts the stale-path timer. The default time is 360 seconds. During this time, Routers A and B continue to forward traffic using the last updated FIB table.
5. Router B begins sending update messages to Router A.
6. Router A starts an update delay timer. Once Router A gets an End-of-RIB (EOR) indication from all of its peers, it starts the BGP route selection process.
7. Router A begins sending update messages, which contain prefix information, to Router B.
8. Router A sends an EOR indication to Router B, so that Router B, in turn, can start its route selection process.
9. Once Router B completes its route selection process, any stale entries in BGP are refreshed with newer information or removed from the BGP RIB and FIB. Router B is now converged.
10. While Router B waits for an EOR, it also monitors the stale-path time. If the timer expires, all stale routes are removed, and normal BGP processes are started.

Note: An EOR marker is an update message with an empty, withdrawn NLRI. It can be used by a BGP speaker to indicate to its peer the completion of the initial routing update after the session is established. For the IPv4 unicast address family, the EOR marker is an update message with the minimum length [BGP-4]. For any other address family, it is an update message that contains only the MP_UNREACH_NLRI attribute [BGP-MP] with no withdrawn routes for that <I, SI>.

## Graceful Restart Operation of BGP with MPLS

The following diagram shows an example in which Provider Edge 1 (PE1) acts as Label Edge Router 1 (LER1), and PE2 acts as LER2.

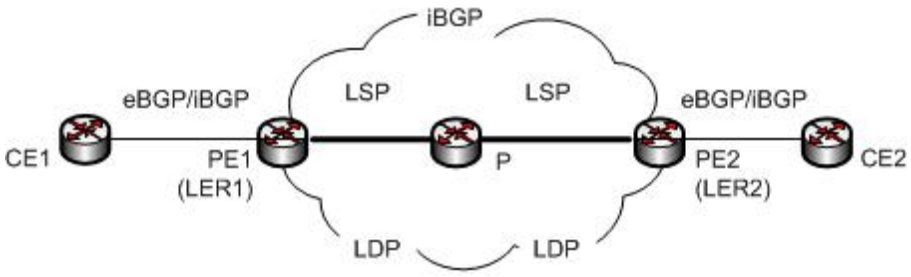



**Figure 9-1: Graceful BGP Restart in MPLS**

The following illustrates and describes the functional flow for the graceful restart operation of BGP with MPLS, assuming the previous example.

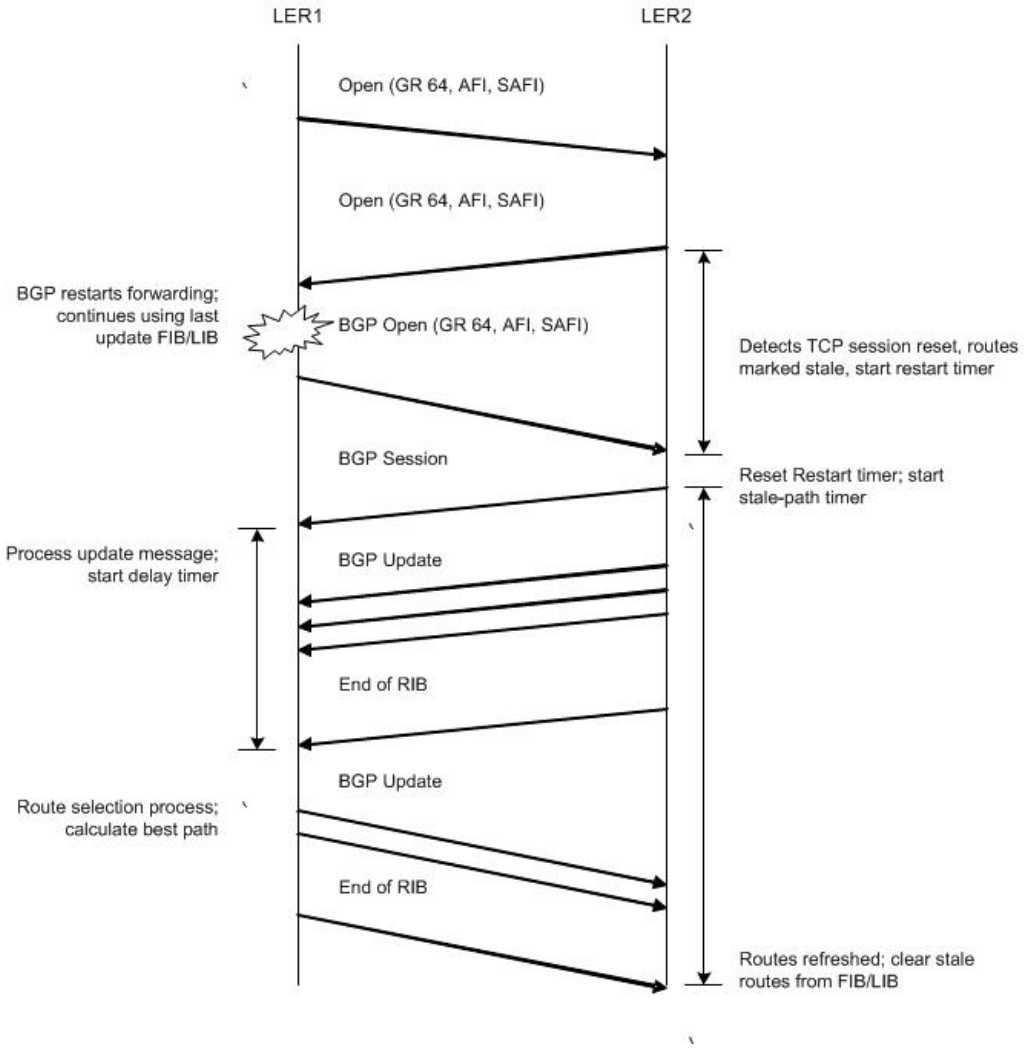



**Figure 9-2: Functional Flow for BGP Graceful Restart in MPLS**

1. LER1 and LER2 exchange BGP graceful restart (GR) information to indicate their capability to preserve the MPLS forwarding state (by including I, SI pairs) during BGP restart: I = 1 and SI = 128.
2. LER1 and LER2 exchange routing information using normal BGP procedures.
3. The BGP process in LER1 restarts.
4. LER2 detects the failure of the BGP session with LER1 and starts the restart timer.
5. LER2 marks the MPLS forwarding state (MPLS VRF and Incoming Label Map [ILM] table information) received from LER1 as stale and starts the stale timer.
6. LER2 stops the restart timer and continues to retain the stale information, if the session is re-established before the restart timer has expired. If the session is not established within the restart time, the stale MPLS forwarding information is immediately deleted.
7. During the session re-establishment process, LER1 sets the restart bit to indicate the BGP restart. LER1 sets the forwarding bit for each address family for which LER2 was able to preserve the MPLS forwarding state across the restart.
8. LER1 rebuilds its RIB and relearns FEC-to-out label mappings. LER2 sends BGP updates to LER1 for each address family. Upon completion of the initial update for an address family, LER2 sends the EOR marker to LER1.
9. LER1 receives BGP Update messages from peers, processes them, and rebuilds its Adj-RIBs-In. However, LER1 defers (configurable delay time) its BGP route-selection process for an address family until the EOR marker for that address family has been received from all peers. After LER1 has selected a route, LER1 updates the Loc-RIB, FIB, MPLS forwarding state, and Adj-RIBs-Out. It then advertises routes to peers.
10. After receiving routing updates from LER1, LER2 replaces and updates the stale MPLS forwarding state. Upon receipt of the EOR marker for an address family from LER1, LER2 deletes any MPLS forwarding state still marked as stale. LER2 uses a (configurable) stale timer.
11. Normal operation resumes.

## Commands Used in Graceful Restart

The following commands are used in graceful restart and graceful reset. For more detailed information, refer to the *Border Gateway Protocol Command Reference*.

**bgp graceful-restart**

Enables BGP graceful-restart capabilities. The default restart time is 120 seconds, and the default stale-path time is 360 seconds.

Command Syntax:

```
[no] bgp graceful-restart (restart-time < 1-3600>|stalepath-time <1-3600>)
```

**neighbor capability graceful-restart**

Configures the router to advertise the graceful restart capability to the neighbors.

Command Syntax:

```
[no] neighbor A.B.C.D|X:X::X:X|EXISTING_PEERGROUP_TAG capability graceful-restart
```

**restart bgp graceful**

Immediately shuts down the router, and notifies NSM it has shut down gracefully and to preserve the routes installed by BGP.

Command Syntax:

```
restart bgp graceful
```

**bgp update-delay**

Specifies the update-delay value for a graceful-restart capable router.

Command Syntax:

```
[no] bgp update-delay <1-3600>
```

**bgp graceful-restart graceful-reset**

In Router mode, invokes graceful restart when a configuration change forces a peer reset.

Command Syntax:

```
[no] bgp graceful-restart graceful-reset
```

### Example

The following shows an example of using the above commands to configure a graceful restart and graceful reset.

```
# configure terminal
(config)# router bgp 100
(config-router)# bgp graceful-restart restart-time 150
(config-router)# bgp graceful-restart graceful-reset
(config-router)# neighbor 1.1.1.1 capability graceful-restart
# restart bgp graceful
```

Note: The entire configuration should be saved before issuing the `restart bgp graceful` command. Also, all of the preceding commands are available only when the configuration option, `--enable-restart`, is enabled when compiling ZebOS-XP. In addition, the `bgp graceful-restart` command must be enabled before the graceful-reset feature, as shown in the following example.

## Enabling Graceful Reset

The following shows an example of using the `bgp graceful-restart` command to enable the graceful-reset feature. The `bgp graceful-restart` command should be executed before enabling graceful reset.

```
# configure terminal
(config)# router bgp 200
(config-router)# bgp graceful-restart
(config-router)# bgp graceful-restart graceful-reset
```

## Events that Cause BGP Peer Reset and Invoke Graceful Restart

All events that cause BGP peer reset (all-session reset) can trigger graceful restart. The following are the valid configuration-change commands that cause a peer reset (all-session reset) and invoke graceful restart:

- bgp router-id A.B.C.D
- no bgp router-id A.B.C.D
- bgp extended-asn-cap
- no bgp extended-asn-cap

The following actions occur as part of a graceful restart:

1. Preserve the ZebOS-XP Loc-RIB information.
2. A flag variable in the route table structure, `struct bgp_table` (`bgpd/bgp_table.h`), preserves of the Loc-RIB information. Value 1 is used to preserve the table, 0 to destroy the table.
3. Store graceful restart information.
4. Store the graceful restart information pertaining to the router/node level, such as, time to restart, time to defer route selection, and the graceful restart state of the node, in variables within the structure, `struct bgp`, in the `bgpd/bgpd.h` files.
5. Set graceful restart properties.
6. Store the graceful restart information pertaining to the peer/neighbor level, such as, time to restart, time to defer route selection, and the graceful restart state of the node, in variables within the structure, `struct bgp_peer`, in the `bgpd/bgpd.h` files.

## Preventing Negative Restart Effects on MPLS Forwarding

ZebOS-XP minimizes the negative effects on MPLS forwarding caused by the Label Edge Router's (LER's) control plane restart. In particular, it handles the restart of the BGP component when BGP is used to carry MPLS labels, and the LER is capable of preserving the MPLS forwarding state across the restart.

With the graceful restart capability enabled, adjacent routers exchange each other's restart capability in their open messages. After that, whenever a router goes down, it preserves its MPLS forwarding table entries. When BGP restarts, minimal or no changes are made to the forwarding table entries, and MPLS forwarding continues uninterrupted. This mechanism ensures that MPLS forwarding remains intact during transient changes in the control plane.

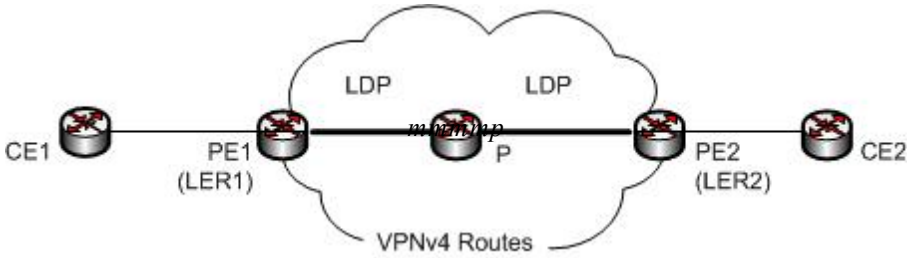



**Figure 9-3: Graceful Restart for Label Edge Routers**

The following capabilities are handled:

- Graceful restart capability for l=1, SI=128.
- Preservation of the MPLS VRF tables and ILM tables at the provider edges during BGP control plane restart.

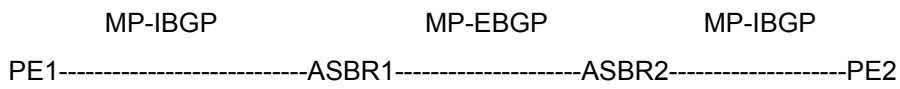
## Configuration Example

```
!
router bgp 100
 bgp graceful-restart                                  (Enables graceful restart capability)
 bgp graceful-restart graceful-reset                   (Enables graceful reset capability)
 neighbor 5.5.5.5 remote-as 100
 neighbor 5.5.5.5 update-source lo
 !
 address-family vpnv4 unicast
 neighbor 5.5.5.5 activate
 neighbor 5.5.5.5 capability graceful-restart (Negotiates VPNv4 graceful restart capability)
 exit-address-family
 !
 address-family ipv4 vrf IPI
 redistribute static
 exit-address-family
!
ip route vrf IPI 21.21.21.21/32 eth1
```

# Handling an Inter-AS Provider

An Autonomous System Border Router (ASBR) must comply with BGP graceful restart for the labeled IPv4 route to continue forwarding for external BGP routes, upon control-plane switchover. It must preserve both incoming and outgoing labels, depending on the options used for inter-AS methods. Because the restarting router preserves both ILM and FTN tables, there is no change during ASBR restart in the logic detailed in the previous sections.

The following figure depicts an inter-AS method, in which an ASBR's ILM table can contain both incoming and outgoing labels.



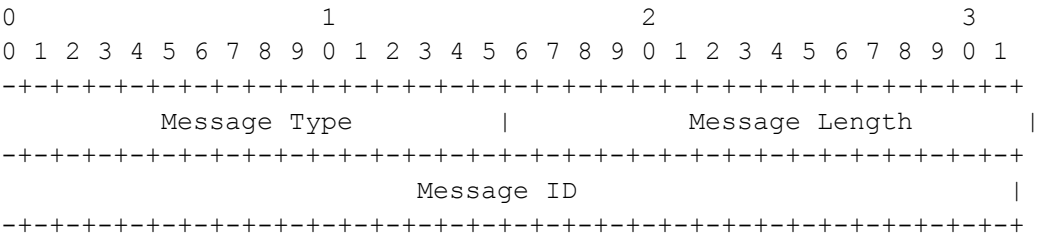

In this case, ASBR1 and ASBR2 act as LSRs, and preserve incoming and outgoing labels. At ASBR1 and ASBR2, VPN labels are swapped. VPN information is transferred over MP-iBGP within each AS, and over single-hop MP-eBGP between the two ASBRs. When an ASBR goes down, it invokes the graceful restart mechanism to preserve the labeled IPv4 routes and updates the labels after restart.

# Recognize Restarting Side Up

The NSM client management can detect whether a client is down or up without any explicit notification from the client.

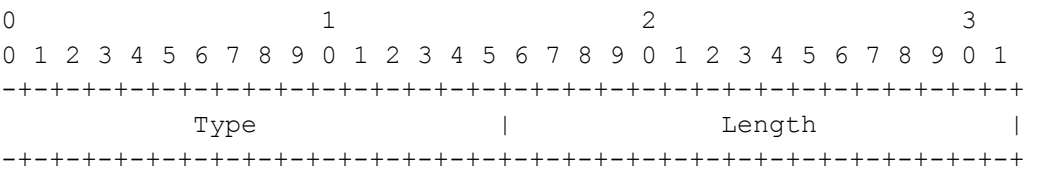
### NSM Message Header

A common NSM message header is defined for each NSM and protocol communication message.

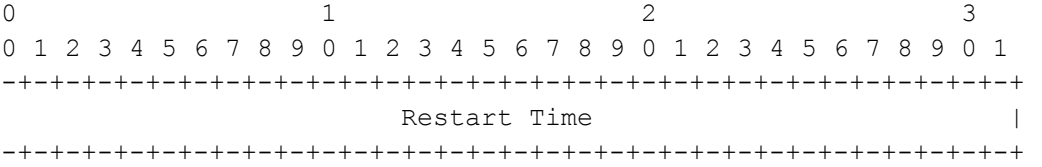



BGP sends the NSM_MSG_ROUTE_PRESERVE message to inform NSM that BGP requires NSM to preserve its routes when BGP goes down gracefully. This message is used to inform NSM to cancel the route preservation when BGP loses the capacity of Graceful Restart.

### NSM TLV format




### Restart Time TLV



NSM_MSG_ROUTE_STALE_REMOVE is issued when BGP does a graceful restart after the BGP daemon restarts and finishes route selection. It requires NSM to remove all retained stale forwarding states of the specified address family.

**NSM TLV format**

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            Type               |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**I/SI TLV**

```
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |              I1               |     SI1     |     Reserved |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

# Re-establish the BGP Session

Use the update message to indicate to its peer the completion of the initial routing update after the session is established. It is very useful for routing convergence.

For an IPv4 unicast address family, it is an UPDATE message with the minimum length.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Unfeasible routes length    | Total path attribute length |
|              (2)              |            (2)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
BGP message header
+ unfeasible routes len
+ total path attribute len
```

For another address family, it is an UPDATE message that contains only MP_UNREACH_NLRI with no withdrawn routes for that <afi, safi>.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Unfeasible routes length    | Total path attribute length |
|              (2)              |            (2)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| BGP_ATTR_FL   | BGP_ATTR_MP_UN| attribute_len |
| AG_OPTIONAL   | REACH_NLRI    |     (1)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              I              |     SI      |
|              (2)            |     (1)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
BGP message header
+ unfeasible routes length
+ total path attribute len
```

```
+ BGP_ATTR_FLAG_OPTIONAL
+ BGP_ATTR_MP_UNREACH_NLRI
+ attribute length
+ I
+ SI
```

Once the session between the restarting speaker and the receiving speaker is re-established, receiving speakers do initial updating, then send out the End-Of-RIB marker to restarting speakers. The restarting speaker receives and processes BGP messages from its peers. However, it defers route selection for an address family, until it receives the End-Of-RIB marker from all its peers (excluding the ones with the Restart State bit set in the receiving capability).

### Configuration Example

```
Restarting side (AS 11)                    Receiving side (AS 33)
         10.10.0.24        ----------------------    10.10.0.32
                                        BGP
Restarting side (10.10.0.24):
      router bgp 11
       bgp graceful-restart time 90
       bgp defer-route-select time 60
       neighbor 10.10.0.32 remote-as 33
       neighbor 10.10.0.32 capability graceful-restart
      !
Receiving side (10.10.0.32):

      !
      router bgp 33
       bgp graceful-restart time 100
       bgp retain-stale time 180
       neighbor 10.10.0.24 remote-as 11
       neighbor 10.10.0.24 capability graceful-restart
```

# CHAPTER 10 Route Server, Filtering, and Load Balancing

ZebOS-XP BGP supports greater control and flexibility for route server, filtering, and load balancing. These features are explained in this chapter.

## Route Server

ZebOS-XP provides many capabilities related to route server, with support for eBGP and iBGP for the IPv4, IPv6, and VPNv4 address families:

| I | SI |
|---|---|
| 1: IPv4 | 1: NLRI for Unicast forwarding (regular IPv4 routes) |
| 2: IPv6 | 1: NLRI for Unicast forwarding (regular IPv6 routes) |
| 1: IPv4 | 128: NLRI for MPLS labeled VPN (used in BGP-MPLS-VPN) |
| 2: IPv6 | 128: NLRI for MPLS labeled VPN (used in 6VPE) |
| 2: IPv6 | 4: NLRI with MPLS labels (used in 6PE) |

These capabilities provide many possibilities:

- A route server can be used to balance loads based on incoming route-updates from a peer. This capability is available for the IPv4 and VPNv4 address families.
- For the VPNv4 address family, the route server does eBGP peering with PE router clients. In certain configurations, there may be several thousand PE routers, making the route server an alternative to full mesh. Also, multiple PEs can belong to the same peer-group.
- Automatic recovery is also supported, using a BGP convergence mechanism when a BGP link fails, or if any other network failure causes preferred routes not to be received at the route server.
- At an Internet Exchange point, many ISPs are connected to each other by external BGP peering. Normally, these external BGP connections are part of a full mesh. As with internal BGP full-mesh formation, this method has a scaling issue. The new capabilities provide a more efficient alternative to this approach.
- A route server can help to resolve this problem, acting almost as a route reflector, except that in practice, it usually uses eBGP connections. In ZebOS-XP, each ISP's BGP router only peers to the route server. However, the route server serves as a BGP information exchange for other BGP routers. By applying this method, the number of required BGP connections is dramatically reduced. The route server also handles inbound and outbound filtering policies for many-to-many eBGP connections, as if each router were connected to many other routers.

## Route Server Example

The following describes dynamic filtering at the route server (per-policy BGP local RIB) and provides an example, based on the following scenario diagram.

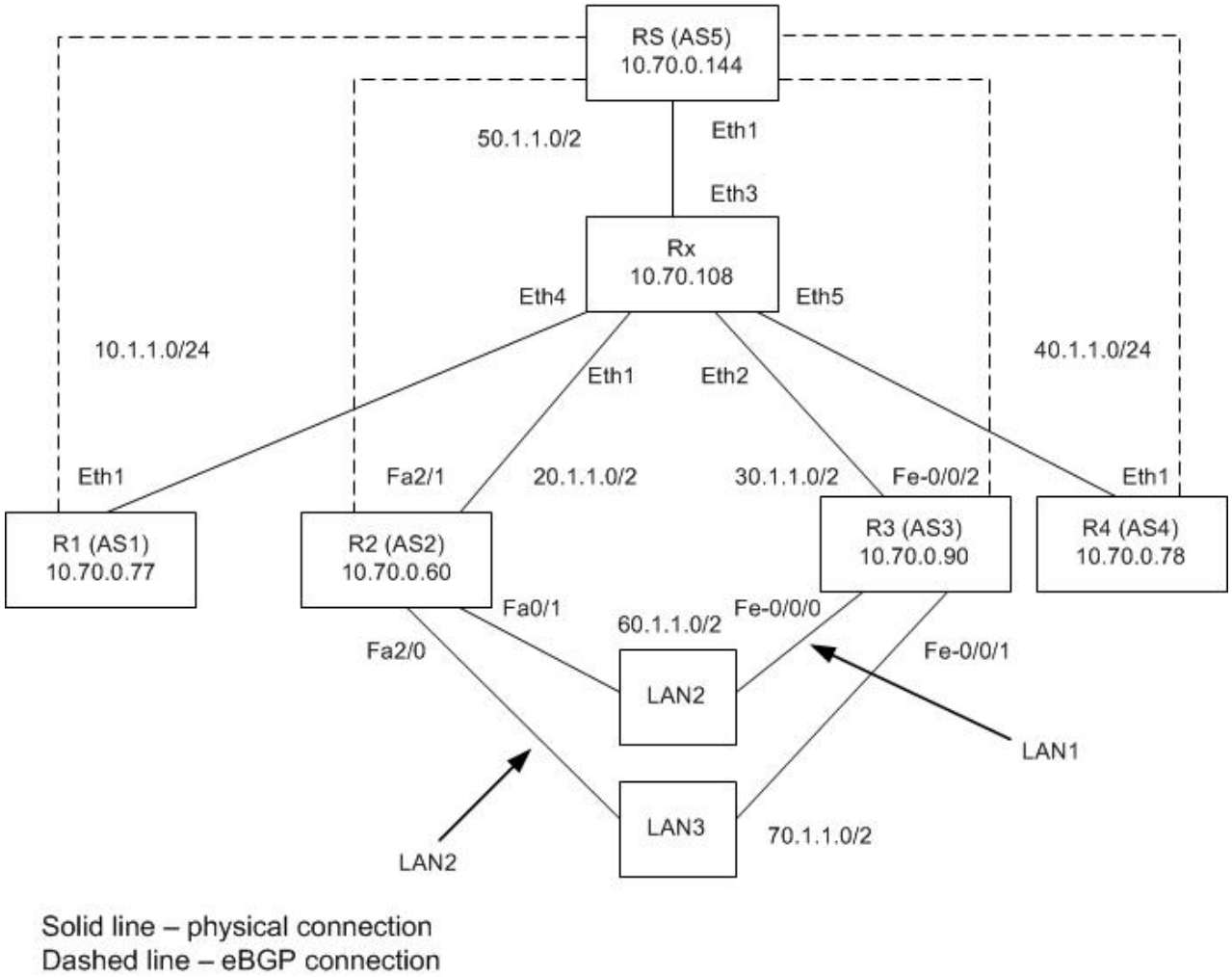



**Figure 10-1: Dynamic Filtering at Route Server**

ZebOS-XP can apply route-map policy X on a set of routes from peer A and route-map policy Y on a set of routes from A, so that policy X-applied routes are preferred. It is also possible to apply one policy when routes are advertised to a peer, and another when routes are advertised to a different peer. The preceding diagram shows a configuration that would benefit from route server.

ZebOS-XP uses a BGP views technique to address this functionality: Instead of a per-peer BGP local RIB, ZebOS-XP maintains a per-policy or per-view local RIB. Thus, a peer can appear in two BGP views, and each view can correspond to different policies. In the network depicted above, route server (RS) peers with R1, R2, R4 and R3. R3 advertises prefixes A and B, while R2 advertises prefixes B and D. RS distributes R and R3 prefixes to R1 and R4.

Two policies are applied in this example:

- Policy 1: When advertising to R1, send prefix A, D and B from R3 (View 1).
- Policy 2: When advertising to R4, send prefix A, D and B from R2 (View 2).

Create two BGP views to accomplish these policies. Each BGP view maintains its own BGP instance and BGP local RIBs, and a route-map is applied for incoming routes.

The following instructions set up a preference for B routes from R3.

**For View 1**

```
Match peer R3
set local-preference 100 for prefix B
```

If, for any reason, R3 does not advertise routes from B, only View 1 will pick B from R2 as the next-hop.

**For View 2**

```
Match peer R2
Set local-preference 100 for prefix B
```

The View 2 instructions cause View 2 to prefer prefix B from R2, but it will pick B from R3 if, for any reason, R2 does not advertise prefix B. The peers that interact with each other by advertising and receiving routes are in the same view.

Note: A peer can also be replaced by a peer group.

It is also possible to choose not to advertise prefix A to R4, or to apply different outgoing route-maps on two different views for applying different filters. This method works for regular IPv4 and IPv6 eBGP/iBGP peering between RS and the RS clients, and also for other I/SI combinations.

## Load Balancing and Filtering

In the ZebOS-XP BGP view implementation, a peer can be seen in more than one view. Use the `bgp multiple-instance allow-same-peer` command.

The following rules apply to filtering and load balancing:

- When a route update (indicating a route is added or deleted) is received, if multiple-instance is enabled with the allow-same-peer option, the route is updated in all views in which the peer is a member.
- When the BGP peer is cleared, it withdraws all routes advertised by this peer in all views.
- When the operator removes a peer from a particular view (using the `no neighbor` command), all routes advertised by the peer are withdrawn from that particular view. The same BGP instance is also removed from the list of BGP processes that peer maintains. If it is the only instance, the peer, itself, is deleted. Similar steps are taken when a view containing a peer associated with other views is deleted.

To support dynamic filtering, the RD_NODE list (for MPLS VPN routes) is maintained on a per-view basis. If a view does not exist, the route is updated in the global RD_NODE list. All of these steps are also supported for VPN routes.

If a route can be reached from two different routers, the operator can choose either of them for the next hop. If the preferred next-hop goes down, the (dynamically calculated) alternative path is automatically used.

For example, Figure 10-1, LAN2 and LAN3 can be reached through R2 or R3. If the operator configures R3 as the preferred next-hop for LAN3, as long as Link H and Link C are both up and running, the traffic to LAN3 is forwarded through R3. If a link failure occurs at either Link H or Link C, traffic is automatically forwarded through R2 using the available links.

ZebOS-XP supports route filtering based on the route source from which the route is learned. In addition, you can allow the same peer in multiple views, so that the peer routes are filtered according to the different policies placed in each view. For example, ZebOS-XP can forward the routes from A to C in one view and forward the routes from A to B in another view. This is achieved using the `match peer` command. The `match peer` command lets ZebOS-XP match all routes or selected routes from a peer in a view, as shown in the following example:

```
route-map in-A permit 10
match peer A.B.C.D
match prefix X.X.0.0/16
set MED 300
```

The preceding instructions set the specified metric and weight for the routes specified in access-list 1 coming from A.B.C.D. If another peer (for example, X.Y.Z.) advertises the same routes, the metric and weight will not be set there. During BGP best-path selection, the route from A.B.C.D will be preferred.

## Commands

The following commands are used in route server, load balancing, and route filtering. For more details on these commands, see the *Border Gateway Protocol Command Reference*.

- `bgp multiple-instance (allow-same-peer)`
  The command `bgp multiple-instance` has been modified. When this command is issued with the `allow-same-peer` option, a default BGP instance is not created. Issuing this command enables the same peer in more than one view. The `no` form of the command disables the allow-same-peer feature. However, the `no` form of the command is not valid if any BPG instances are present.
- `show ip bgp vpnv4 view WORD all`
  This command displays all IPv4 VPN routes learned in a view.
- `show ip bgp vpnv6 view WORD all`
  This command displays all IPv6 VPN routes learned in a view.
- `match ip peer (<1-199>|<1300-2699>|WORD)`
  This command matches the peer address of the route. It applies policies based on the route source. The BGP TCP/IP session is formed using the specified address, unlike the next hop in the update message. The `no` form of the command disables the policies based on the route source. The IPv6 form of this command is `match ipv6 peer (<1-199>|<1300-2699>|WORD`.

## Route Server Best-path

In the example depicted below, the route server selects the best-path on a per-peer/per-peer-group basis for IPv4 VPN-Unicast routes, with overlapping Route Distinguishers.

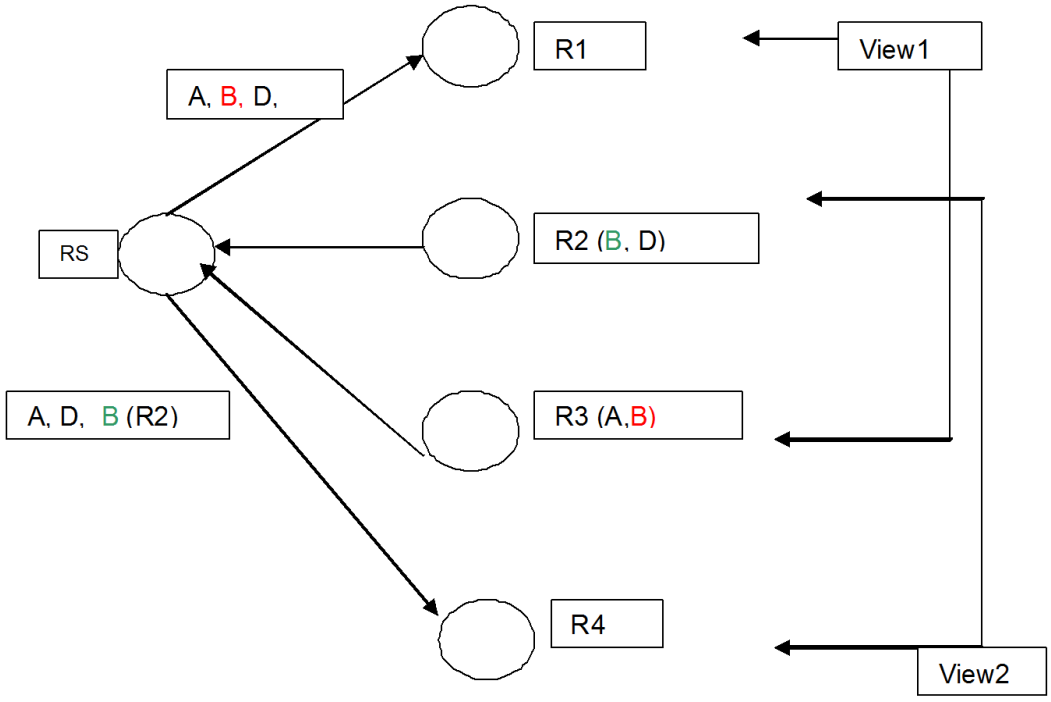



**Figure 10-2: Router server best-path selection**

In this case, Routers 1 through 4 are PE routers of the red VPN. All share the same route distinguisher. R1 and R4 have local-loopback interfaces in the red VPN, and R2 and R3 have the LAN_02 interface in the red VPN. Routers 2 and 3 both advertise LAN02 as a VPN (LAN_O2_VPN) route and LAN3 as a regular IPv4 unicast route. Standard BGP best-path selection on the route server would normally select the lowest BGP peer ID as the active route. Best-path selection for LAN_02_VPN is handled differently from that for R1 and R4. This is achieved in the following sequence:

1. R1 learns from the route server (RS) that R2 is the best path for LAN_02_VPN, and R3 is the best path for LAN_03.
2. R4 learns from RS that R3 is the best path for LAN_02_VPN, and R2 is the best path for LAN_03.
3. R2 is triggered to withdraw both LAN_02_VPN and LAN_03 reachability. Then, RS updates both R1 and R4, informing them that R3 is the best path to LAN_02_VPN and LAN_03.
4. R2 is then triggered to re-advertise LAN_02_VPN and LAN_03 to RS. RS then updates R1 and R4, as described in Steps 1 and 2, above.
5. R3 is now triggered to withdraw both LAN_02_VPN and LAN_03 reachability, at which point RS updates both R1 and R4, informing them that R2 is the best path to LAN_02_VPN and LAN_03.
6. R3 is then triggered to re-advertise LAN_02_VPN and LAN_03 to RS. RS then updates R1 and R4, as described in Steps 1 and 2, above.
7. Link B is disabled on R2. RS detects the BGP session time-out and updates R1 and R4.
8. Link B is enabled on R2. BGP is re-established between R2 and RS. RS re-advertises to R1 and R4, as described in Step 4.
9. Steps 6 and 7 are repeated with Link C and R3.

# Weight Command

Through the `neighbor weight` command, different weights can be assigned to different address families of a peer.

This feature enhances control and flexibility in specifying preferred routes. For example, the system can be configured to prefer VPNv4 routes from peer A and IPv4 routes from peer B. This feature is supported for the IPv4, IPv6, VPNv4, VPNv6 and 6PE address-families.

- The `neighbor weight` command assigns a weight to a neighbor connection. All routes learned from this neighbor will initially have the assigned weight. The negation of the command removes the configured value.
- When the above command is given in an address-family mode, it sets the peer weight for that specific address family.
- If address family is not specifically set, the weight is updated for the default address-family.
- If a peer has multiple sessions with a peer, IPv4 and VPNv4, the weight can be specified separately for both cases.
- The `show ip bgp neighbors` command shows the weight and address family.
- The default address family is IPv4.

## Using the weight Command

The value of the weight attribute is used to assign a best path. The weight is assigned locally to each individual router, and the value is only understood by that particular router. The value is neither propagated nor carried through in route updates.

A weight can be any integer from 0 to 65,535. Routes learned through another BGP peer have a default weight of 0, and routes sourced by the local router—the paths that the router originates—have a default weight of 32768. Routes with a higher weight value have preference when there are multiple routes to the same destination.

If a BGP peer group is specified using the peer-group-name argument, all of the members of the peer group will inherit the characteristic configured with this command.

The `no` form of this command removes the command from the running configuration. However, route selection is not done on the basis of the default weight, because the default weight that should be applied when the weight is unconfigured is only applied to the fresh routes, but not to the previously present routes. To apply the default weight, use either a soft reset command or the `clear ip bgp peer-group` command.

The weights assigned with the `set weight` command override the weights assigned using the `neighbor weight` command.

# Distance Command

Administrative distance is a dependability rating for the source of routing information. Higher distance values imply lower trust ratings. BGP uses three different administrative distances: internal, external, and local. Routes learned through internal BGP (iBGP) are given the internal distance, routes learned through external BGP (eBGP) are given the external distance, and routes learned through IGP and the static routes that are redistributed are given the local distance.

Use the `distance` command to configure the administrative distance associated with routes.

Distance can be configured based on the address family. The `distance` command can be used to help BGP select a better route to a node than the route selected by BGP by default. This command lets the user configure BGP to prefer certain internal routes over external routes.

## Using the distance Command

To set an administrative distance, specify a value for all three of the following types of routes:

- External: the administrative distance for BGP external routes. External routes are routes for which the best path is learned from a BGP peer external to the AS. Acceptable values are from 1 to 255. The default value is 20.
- Internal: the administrative distance for BGP internal routes. Internal routes are routes learned from a BGP peer within the same AS. Acceptable values are from 1 to 255. The default value is 200.
- Local: the administrative distance for BGP local routes. Local routes are those routes locally originated by BGP. BGP can locally originate routes if the `network` command is issued, if redistribution into BGP is configured, or via a non-AS-set aggregate route. Acceptable values are from 1 to 255. The default value is 200.

When the `no distance` command is given, the values of distance_ebgp, distance_ibgp, and distance_local variables are set to 0. If the distance value is 0, the default distance (eBGP - 20, iBGP/local - 200) will be applied based on the type of route.

A value of 255 indicates the routing information is not trusted, and the information is not installed in the routing table.

The `no distance` command can be used to return to the default values.

The following APIs handle this enhanced capability:

- `bgp_distance_config_set` sets the distances configured either in Address Family or Router mode.
- `bgp_distance_config_unset` clears the distances configured for that Address-Family mode. If this API is called when distances are cleared in the Router mode, only IPv4 unicast distances are cleared.

Note: For previous users of the `distance` command, the only change in its use is that they can be employed in different modes for a variety of supported address-families.

The following provides examples of configuring the enhanced BGP `distance` command in different modes, and outputs using the `show running-config` command.

### Example: Configure BGP Distance

The following configures the command in Router mode:

```
(config)#router bgp 10
(config-router)#distance bgp 10 30 40
```

Output:

```
router bgp 10
distance bgp 10 30 40
```

### Example: Configure Distance for Targeted I

The following configures the command for a particular I, in this case, IPv6:

```
(config-router)#address-family ipv6
(config-router-)#distance bgp 10 123 120
```

Output:

```
router bgp 10
distance bgp 10 30 40
 !
 address-family ipv6
 distance bgp 10 123 120
exit-address-family
```

Note: In the above output, only the configured address family is displayed.

### Example: Remove Distance Configuration for Router Mode Setup

The following unconfigures the command in Router mode:

```
(config)#router bgp 10
(config-router)#no distance bgp
```

Output:

```
router bgp 10
 !
 address-family ipv6
 distance bgp 10 123 120
 exit-address-family
```

Note: In the above output, the address family is displayed because the distance is cleared in Router mode.

### Example: Remove Distance Configuration for Instance Setup

After configuration, to unconfigure the distance, use the command in IPv6 mode:

```
(config-router)#address-family ipv6
(config-router)#no distance bgp 10 123 120
```

Output:

```
router bgp 10
```

In the above example, the output shows that the address-family distance is removed. However, if the command is executed, as shown in the previous example, if the address-family distance is not already configured, the Router mode distance will be applied. Otherwise, the default distance will be applied.

### Example: Remove Distance Settings Configured in Router Mode

The following example is used if the distance is not configured in the Address-family mode, and the `no` form of the command is used in Router mode:

```
(config)#router bgp 10
(config-router)#no distance bgp 24 120 150
```

Output:

```
router bgp 10
```

### Example: Configure Distance in IPv6 Mode

The following configures the command for a particular I, in this case, IPv6:

```
(config)#router bgp 10
(config-router)#address-family ipv6
(config-router-)#distance bgp 24 120 150
```

Output:

```
router bgp 10
 !
 address-family ipv6
 distance bgp 24 120 150
 exit-address-family
```

**Example: Remove Distance Settings Configured in IPv6 Mode**

The following unconfigures the distance using a particular Address-family mode:

```
(config)#router bgp 10
(config-router)#address-family ipv6
(config-router-)#no distance bgp 24 120 150
```

Output:

router bgp 10

# CHAPTER 11 Inter-AS BGP MPLS VPNs

A virtual private network (VPN) is a network in which customer connectivity to multiple sites is deployed on a shared infrastructure, called the backbone, to provide the same administrative policies and security as a private network.

A VPN consists of a set of sites connected by the MPLS core backbone. Two sites communicate over the backbone when they have at least one VPN in common between them.

Each VPN site must contain one or more customer edge (CE) devices. Each CE device is attached, by an attachment circuit, to one or more provider edge (PE) routers. Routers in the service provider's (SP) network that do not attach to CE devices are called P routers.

If every router in an SP's backbone had to maintain routing information for all VPNs supported by the SP, there would be severe scalability problems; the number of sites that could be supported would be limited by the amount of routing information that could be held in a single router. Therefore, it is important that the routing information about a particular VPN is only required to be present in the PE routers that attach to that VPN. For this reason, BGP is a good protocol for distributing MPLS labels.

When two sites of a VPN are connected to different autonomous systems (ASs), (because the sites are connected to different SPs or the SP's different ASs), the PE routers attached to that VPN will then be unable to maintain Internal BGP (iBGP) connections with each other, or with a common route reflector. Instead, there must be a way to use External BGP (eBGP) to distribute VPN-IPv4 addresses.

## Inter-AS BGP MPLS VPN Solution

The extension of the MPLS VPN functionality allows:

- PE routers in a provider backbone to be in different ASs, instead of requiring them to belong to the same AS
- BGP to exchange routes between eBGP and iBGP
- Configuration of an eBGP neighbor to be a VPN peer
- Extension to IPv6 on PE (6PE) and IPv6 on VPN to PE (6VPE) routers for inter-provider scenarios

### BGP Route Exchange Between eBGP and iBGP

BGP can exchange routes between eBGP and iBGP. Previously, eBGP VPN routes were only sent to a peer if the router was a route reflector: this is no longer the case.

BGP also allows the exchange of BGP-MPLS-VPN routes received from peers to other capable peers.

Negotiation of VPN-IPv4 (and/or VPN-IPv6 and 6PE) capability can be configured on eBGP peers. Once the capabilities are negotiated, the iBGP routes learned from the iBGP peer PEs are then exchanged, using multiprotocol-BGP, to the eBGP VPN capable peer. On the eBGP peer, the routes learned from eBGP are then exchanged with iBGP VPN capable peers.

Consider the following case:

#### eBGP redistribution of labeled VPN-IPv4 routes from AS to neighboring AS

The PE routers use iBGP to redistribute labeled VPN-IPv4 (and/or VPN-IPv6, 6PE IPv6) routes to either an Autonomous System Border Router (ASBR), or to a route reflector of which an ASBR is a client. The ASBR then uses

eBGP to redistribute these labeled VPN-IPv4 routes to an ASBR in another AS, which in turn, distributes them to the PE routers in that AS, or to another ASBR, which in turn distributes these routes.

## eBGP Neighbor Can Be VPN Peer

BGP allows explicitly configuring an eBGP neighbor to be a VPN peer using the CLI, `neighbor <IPv4 or IPv6 address> allow-ebgp-vpn`. When this configuration is done, negotiation of the eBGP VPN is allowed.

eBGP connection between ASs multiple hops away can be used to negotiate VPN capability.

In this case, the default behavior is to allow eBGP multihop. After the capabilities are negotiated, the VPN routes are then exchanged between the eBGP peers.

Consider the following case:

Multi-hop eBGP redistribution of labeled VPN-IPv4 routes between source and destination ASs, with eBGP redistribution of labeled IPv4 routes from AS to neighboring AS.

There is a direct eBGP interaction between the PE routers, and they exchange VPN-IPv4 (and/or VPN-IPv6, 6Pe IPv6) routes directly without any interaction with the ASBR.

In turn, the ASBR must be configured to redistribute PE host routes to other eBGP peers, and from there, into the IGP in the peer.

# CHAPTER 12 BGP MPLS for VPN

ZebOS-XP supports BGP-MPLS VPNs for IPv4 and IPv6. This lets service providers use an IP backbone to provide IPv4 and IPv6 VPN connectivity to their customers. Customer data packets are tunneled through the backbone, so that core routers are unaware of IPv4 and IPv6 VPN routes.

The BGP module supports IPv4 and IPv6 VPN route exchange, and NSM supports IPv4 and IPv6 FTN and ILM tables required to tunnel customer data.

All commands specific to IPv4 VRF are applicable to IPv6 VRF, and all commands specific to the VPNv4 address-family are applicable to the VPNv6 address-family.

ZebOS-XP also supports Provider Edge routers (PEs) in multi autonomous system (multi-AS) backbones, as follows:

- eBGP redistribution of labeled VPN-IPv6 routes from an AS to a neighboring AS.
- Multihop eBGP redistribution of labeled VPN-IPv6 routes between source and destination ASs, with eBGP redistribution of labeled IPv6 routes from an AS to a neighboring AS.

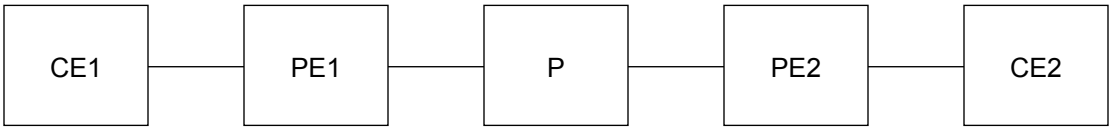
## Features

ZebOS-XP supports BGP MPLS for VPN features listed below.

- BGP CLI modes for VPNv4 and VPNv6, and IPv4 and IPv6 VRF.
- CLIs that allow IPv4 and IPv6 address-family configuration.
- BGP show commands for IPv4 and IPv6, with distinguishing based on I and SI values.
- BGP VFR configuration based on I and SI for IPv4 and IPv6.
- BGP peer activation based on VPNv4 and VPNv6 address families.
- BGP OPEN and UPDATE messages to accommodate VPNv4 and VPNv6 based Network Layer Reachability Information (NLRI) capabilities, and IPv4 and IPv6 link local and global address-based next-hops.
- Next-hop encoding for BGP speakers requesting IPv4 or IPv6 transport.
- NSM ability to add static routes into IPv4 or IPv6 VRF, and corresponding show commands.
- Separate FTN and ILM tables for IPv4 and IPv6 in NSM.

## Configuration Example

The following is a sample configuration for PE1 and PE2 as iBGP neighbors using the topology below.




### PE1: PE with VPNV6 address family on IPv6 transport

```
configure terminal
router bgp 100
bgp router-id 13.13.13.13
neighbor 3ffe:15:15:15:15::0 remote-as 100
neighbor 3ffe:15:15:15:15::0 update-source 3ffe:13:13:13:13::0
address-family vpnv6 unicast
neighbor 3ffe:15:15:15:15::0 activate
```

### PE1: Configuring VRF and enabling interface

```
ip vrf IPI
interface eth1
ip vrf forwarding IPI
```

### PE1: Configuring route distinguisher (RD) and route target (RT) for VRF IPI

```
ip vrf IPI
rd 1:100
route-target both 100:200
```

### PE2: PE with VPNV6 address family

```
configure terminal
router bgp 100
bgp router-id 15.15.15.15
neighbor 3ffe:13:13:13:13::0 remote-as 100
neighbor 3ffe:13:13:13:13::0 update-source 3ffe:15:15:15:15::0
address-family vpnv6 unicast
neighbor 3ffe:13:13:13:13::0 activate
```

### PE2: Configuring VRF and enabling interface

```
ip vrf IPI
interface eth2
ip vrf forwarding IPI
```

### PE2: Configuring RD and RT for VRF IPI

```
ip vrf IPI
rd 1:200
route-target both 100:200
```

# CHAPTER 13 IPv6 Islands over MPLS Using 6PE

IPv6 Provider Edge (6PE) routers are used to connect IPV6 islands over an IPv4 MPLS cloud. 6PE routers are dual stack, in order to connect to IPv6 islands and MPLS core, which is only required to run IPv4 MPLS. 6PE routers exchange IPv6 reachability information transparently over the core using the multiprotocol BGP over IPv4. The BGP Next-hop field conveys the IPv4 address of the 6PE router, so that dynamically established IPv4-signaled MPLS LSPs can be used without explicit tunnel configuration.

## Features

ZebOS-XP IPv6 Islands over MPLS Using 6PE features are listed below.

- `HAVE_6PE` compilation flag to enable 6PE functionality.
- BGP command mode for 6PE (labeled-unicast mode).
- Command to allow address family configuration.
- BGP show commands for 6PE; distinguishing based on I and SI values.
- BGP peer activation based on labeled-unicast address family.
- BGP OPEN/UPDATE messages to accommodate 6PE based NLRI capabilities.
- Next-hop encoding for BGP speakers requesting IPv6 transport in an IPv4 core.

## HAVE_6PE Compilation Flag

The `HAVE_6PE` flag is enabled only when the `HAVE_IPV6` and `HAVE_MPLS` flags are enabled.

The 6PE related code is enclosed within the `HAVE_6PE` flag and the 6PE functionality is enabled only when the three flags, `HAVE_6PE`, `HAVE_IPV6`, and `HAVE_MPLS`, are enabled.

## BGP_6PE_process Function

This function handles both the IPv6 and 6PE routes, when the 6PE flag is enabled, and the peer flag is set.

A check for the peer flag, PEER_FLAG_6PE_ENABLED, is done: if enabled, the SI value is changed, and the value is sent to the `bgp_announce_check` function for ILM installation, if required.

FTN installation of the routes is done, and the flag BGP_INFO_6PE_FTN_INSTALLED is set for the 6PE route.

# CHAPTER 14 BGP Dynamic Capability

BGP Dynamic Capabilities are protected names. A capability designates something within a computing system. They cannot be depicted because they are not composed of characters. Capability systems have message-passing primitives that allow capability holders to send capabilities in messages to other programs. While directories usually have protection structures that define content accessibility, capabilities just access the contents. So, when a program passes a capability to another, in essence, it passes its authority to the other program.

## Overview

The BGP Dynamic Capability function allows a holder to dynamically change their capability value. It enables an administrator to change the address family configuration after a BGP connection is established. Normally, BGP capabilities are only advertised in the OPEN message during the session initialization. To add or remove a certain capability, such as Address Family Support, the session in progress must be reset, which might disrupt other services running over the session. Dynamic Capability permits the update of capabilities over a session already in progress.

## Considerations

A Capability message contains one or more of the following tuples:

Action (1 octet)

Capability Code (1 octet)

Capability Length (1 octet)

Capability Value (variable)

The value of the Action field is 0 for advertising a capability, and 1 for removing a capability.

ZebOS-XP uses the following values for capability code and message:

- Capability Code: 3
- Capability Message: 6

### neighbor capability dynamic

This command enables the dynamic capability for a specific peer. This feature is disabled by default. When a user specifies this option, the address family configuration can be changed without stopping BGP connection.

#### Command Syntax

```
(no) neighbor IPADDRESS capability dynamic
```

#### Parameters

`IPADDRESS = (A.B.C.D|X:X::X:X)` The address of the peer for which dynamic capability is set.

## Testing

To test this feature, change the address family configuration.

First:

```
!
router bgp 1
 network 10.0.0.1
 neighbor 10.0.0.1 remote-as 1
 neighbor 10.0.0.1 capability dynamic
!
 address-family ipv6 unicast
 network 3ffe:506::/32
 neighbor 10.0.0.1 activate
 exit-address-family
!
```

Then, remove IPv6 configuration:

```
conf t
router bgp 1
address-family ipv6 unicast
no neighbor 10.0.0.1 activate
```

The remote side does not receive any IPv6 prefix (3ffe:506::/32), and the peer is still established.

# CHAPTER 15 Outbound Routing Filter

This chapter contains information about the Outbound Routing Filter (ORF) feature in ZebOS-XP.

## Introduction

ORF exchanges filtering data among routing peers that each implements, on behalf of the others, to block certain routing updates. This exchange is accomplished when routers and peers advertise, and respond, to advertisements of ORF capabilities. When routers detect ORF capabilities from peers, the routers push ORF inbound prefix filters that are then installed by the peers as outbound filters.

BGP ORF capability has three configurable modes:

- Send
- Receive
- Both

To apply the ORF mechanism, a local router advertises the ORF capability in Send mode to indicate it has ORF entries (including a set of prefix-list filters) to send to a remote peer. The remote peer advertises the ORF capability in Receive mode to indicate it can receive ORF entries from its peers. After receiving the route updates from the remote peer, the local router applies its inbound prefix list to those updates.

Several advantages result from this technology:

- The local BGP speaker does not consume resources to generate the unnecessary routing updates that the neighbor filters during input.
- Less work for the remote BGP speaker by handling less route updates.
- Link bandwidth usage is decreased due to less sending of unnecessary routing updates.
- The ability to configure many neighboring routers from this central route reflector.

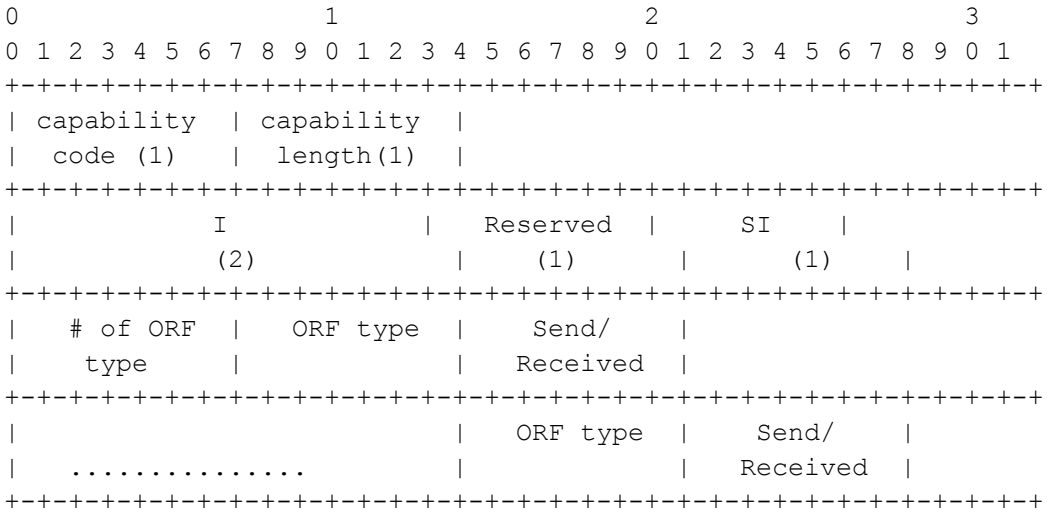
When a BGP session is established, ZebOS-XP checks for ORF capability support; if ZebOS-XP finds this support, it sends the REFRESH_REQ message, then sends an industry-standard KEEPALIVE message. It is possible for both messages to be simultaneously sent through one TCP buffer.

# Detailed BGP ORF Mechanism

## Cooperative Route Filtering Capability

### Encoding

A BGP speaker capable of receiving ORF entries from its peers, or a BGP speaker that has ORF entries to send to its peers, advertises these capabilities to peers using the Cooperative Route Filtering Capability.




where:

Capability code: 3

Capability length: variable

ORF type: ORF_TYPE_PREFIX or ORF_TYPE_PREFIX_OLD

Send/Receive:

willing to **receive** ORF entries from its peer (value 1)

would like to **send** ORF entries to its peer (value 2)

**both** (value 3)

### Capability Configuration

Use this command to configure the ORF capability. The default for this capability is OFF. BGP ORF support is available for IPv4 Unicast, IPv4 Multicast, and IPv6 Unicast.

Command syntax

```
neighbor (A.B.C.D|X:X::X:X|WORD) capability orf prefix-list (send|receive|both)
```

| | |
|---|---|
| `send` | The local router would like to send ORF entries to its peer. Individual routers or peer groups can be configured in Send mode. The peer-group members cannot be configured in Send mode. |
| `receive` | The local router is capable of receiving ORF entries from its peer. Only individual routers or peer groups can be configured in Receive mode. The peer-group member cannot be configured in Receive mode. |
| `both` | The local router can send ORF entries to its peer and can also receive ORF entries from its peer. Only an individual router or peer group can be configured as Receive mode. The peer-group member cannot be configured as Receive mode. |

Command mode:

BGP router configuration

BGP router address-family configuration

Example:

```
neighbor 10.10.0.5 capability orf prefix-list send
```

### Capability Negotiation

To apply the ORF mechanism, the local router advertises the ORF capability in Send mode, which indicates the capability to send ORF entries (including a set of prefix-list filters) to its peer. The remote peer advertises the ORF capability in Receive mode, which indicates the capability to receive ORF entries from peers.

ORF capability information can be sent with the BGP OPEN message when BGP neighbors negotiate, and can be sent with the CAPABILITY message if the BGP router supports dynamic capability.

## Execute Cooperative Route Filtering Procedure

### Local Speaker

After the ORF capability negotiation, the local speaker sends the BGP ROUTE_REFRESH message to BGP peers: this message carries a set of Outbound Route Filters (ORFs). The peers apply these filters, in addition to locally configured Outbound filters, to constrain and filter outbound routing updates to the speakers.

Execution of the following command sequence pushes over a prefix list and receives route refresh from the remote BGP router:

1. clear ip bgp * in prefix-filter
2. clear ip bgp * ipv4 (unicast|multicast) in prefix-filter
3. clear ip bgp A.B.C.D in prefix-filter
4. clear ip bgp A.B.C.D ipv4 (unicast|multicast) in prefix-filter
5. clear ip bgp peer-group WORD in prefix-filter
6. clear ip bgp peer-group WORD ipv4 (unicast|multicast) in prefix-filter
7. clear ip bgp external in prefix-filter
8. clear ip bgp external ipv4 (unicast|multicast) in prefix-filter
9. clear ip bgp <1-65535> in prefix-filter
10. clear ip bgp <1-65535> ipv4 (unicast|multicast) in prefix-filter

When the inbound prefix list changes (or is removed), these commands can be used to push out the new prefix list and receive the router refresh based on the new prefix list.

**Syntax Description**

| | |
|---|---|
| * | all the peers |
| A.B.C.D | Address of the neighbor |
| WORD | BGP peer-group name |
| external | all the eBGP peers |
| ipv4 (unicast\|multicast) | the specified address family |
| <1-65535> | all the peers belong to this AS |

### Remote BGP Speaker

The incoming ROUTE_REFRESH message specifies an <I, SI, ORF_TYPE> tuple that a peer has capability of, or requires filtering.

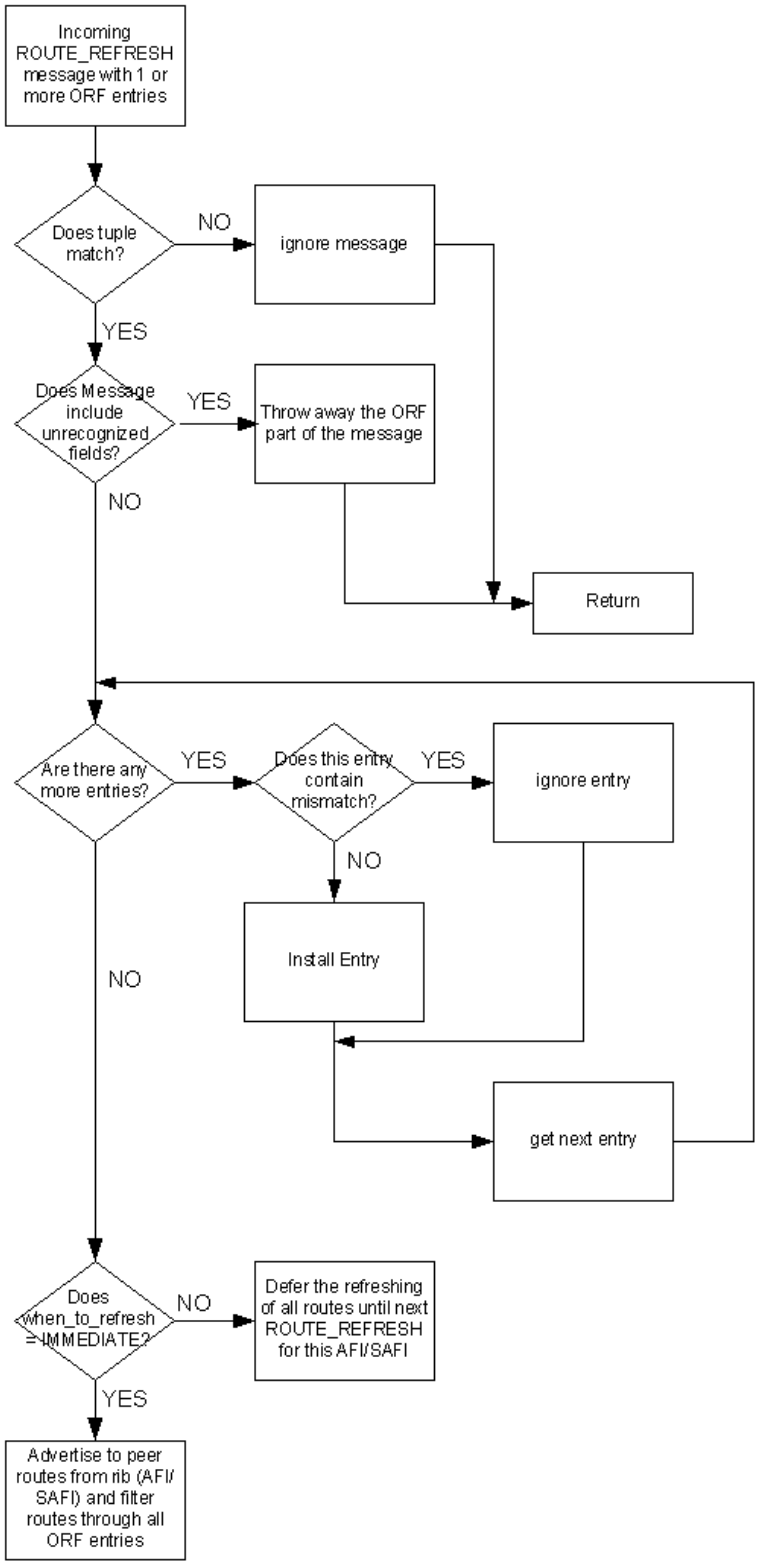



**Figure 15-1: Process Flow for Remote BGP Speaker**

If the tuple matches what the router requires, or has capability of, processing continues.

Process the ORF entries one by one, until none remain unprocessed. If none exist to start with, continue.

The BGP peer does not advertise any route updates to the local router, until it receives the ROUTE_REFRESH request from the local router with the when-to-refresh set to IMMEDIATE.

## Related Commands

Querying commands to display the prefix list received from a neighbor:

```
show ip bgp neighbors (A.B.C.D|X:X::X:X) received prefix-filter
show ip bgp ipv4 (unicast|multicast) neighbors (A.B.C.D|X:X::X:X) receive prefix-filter

  syntax description:
   ------------------------------------------------------------
  | (A.B.C.D | X:X::X:X)      |  neighbor ipv4 /ipv6 address   |
  -------------------------------------------------------------
  | ipv4 (unicast| multicast) |  specified address-family      |
   ------------------------------------------------------------
```

# CHAPTER 16 Multiple Instances

With the ZebOS-XP BGP Multiple Instance feature, two or more independent routing entities (instances) can exist inside a single BGP process or task.

To implement this feature, ensure that each BGP instance supports:

- A connection with different BGP networks without exchanging BGP routing information.
- No routes installed in the NSM RIB or kernel FIB.
- All BGP peers, configured in any of the BGP instances, operate in a single FIB.

## Benefits

BGP shares all routing information with all peers without any outbound policy configuration. In certain cases, for example, it is not desirable:

- For the unstable AS to affect another AS in the other side
- For each side to exchange BGP routes with each other for a certain reason
- To flush BGP routes to the other side to cause heavy traffic in each side

It is required to isolate all of these BGP connections without sharing any BGP routes with each other, but expect the forwarding not to be affected.

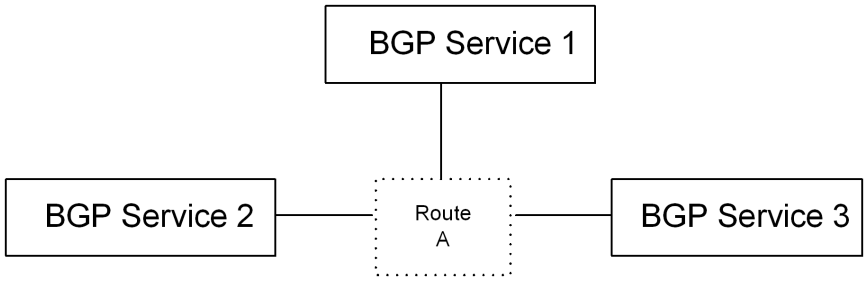



**Figure 16-1: BGP Multiple Instance Example**

## Basic BGP Data Structure

Each BGP instance consists of a:

- Unique instance name (the default instance does not require a name).
- Set of routing tables.
- Set of peers and peer groups.
- Set of routing option configuration (for example: BGP feature configuration for each instance).
- BGP master that keeps a record of BGP instances and does global configuration for each BGP instance. For example, the master guarantees a unique instance name for each BGP instance.

For details about BGP data structures, refer to Chapter 18, *BGP Data Structures*

## Interaction with Other Components

- All instances share common interface information received from NSM.
- Each instance can send redistribution requests (routing updates) to NSM.
- NSM sends the required redistribution information back to the BGP daemon, flushing the instance.
- The instance, through redistribution configuration, checks to get the information it requires.

## Commands for Multiple Instances

The *Border Gateway Protocol Command Reference* describes these commands.

### Create / delete BGP instances

router bgp <1-65535> view WORD

no router bgp <1-65535> view WORD

### Query instance information

show ip bgp view WORD neighbor

show ip bgp view WORD neighbor (A.B.C.D|X:X::X:X)

show ip bgp view WORD summary

show ip bgp view WORD ipv4 (unicast|multicast) summary

show bgp view WORD summary

show bgp view WORD ipv6 summary

### Reset BGP session with all peers of specified BGP instance

clear ip bgp view WORD *

clear bgp view WORD *

### Soft reconfigure BGP peers

clear ip bgp view WORD * soft out

clear ip bgp view WORD * ipv4 (unicast|multicast) soft out

clear bgp view WORD * soft out

clear ip bgp view WORD * soft in

clear ip bgp view WORD * ipv4 (unicast|multicast) soft in

### BGP Outbound Routing Filter (ORF) support

clear ip bgp view WORD * in prefix-filter

clear ip bgp view WORD * ipv4 (unicast|multicast) in prefix-filter

# CHAPTER 17 BGP Multipath

## Introduction

Border Gateway Protocol (BGP) routers typically receive multiple paths to the same destination. The BGP best path algorithm decides which is the best path to install in the IP routing table and to use for traffic forwarding.

BGP Multipath allows installation into the IP routing table of multiple BGP paths to the same destination. These paths are installed in the table together with the best path for load sharing. BGP Multipath does not affect bestpath selection. For example, a router still designates one of the paths as the best path, according to the algorithm, and advertises this best path to its neighbors.

The BGP multipath feature is useful for load-balancing on the forwarding path when there are multiple “equal” paths are available for a given prefix. The feature supports both IPv4 and IPv6 and is available for both eBGP and iBGP. Turn on Multipath at NSM for the maximum number of paths desired. BGP allows up to 64 multipaths in eBGP or in iBGP.

### BGP Multipath Overview

ZebOS-XP sends multiple equal-cost multi-path routing (ECMP) requests to NSM after determining ECMP BGP next-hops. This feature is not enabled by default in BGP. Regardless of multipath BGP configuration, BGP sends the best-path route in the UPDATE message as specified in RFC 4271. In ZebOS-XP, a separate configuration knob for BGP multipath numbers is provided. Turn on and configure both BGP multipath and NSM multipaths to use the BGP multipath feature.

Actual multipath is installed in the following order of preference:

FIB multipath number → NSM multipath number → BGP multipath number.

Figure 17-1 depicts an example of multipath-router configured for both iBGP and eBGP load balancing.

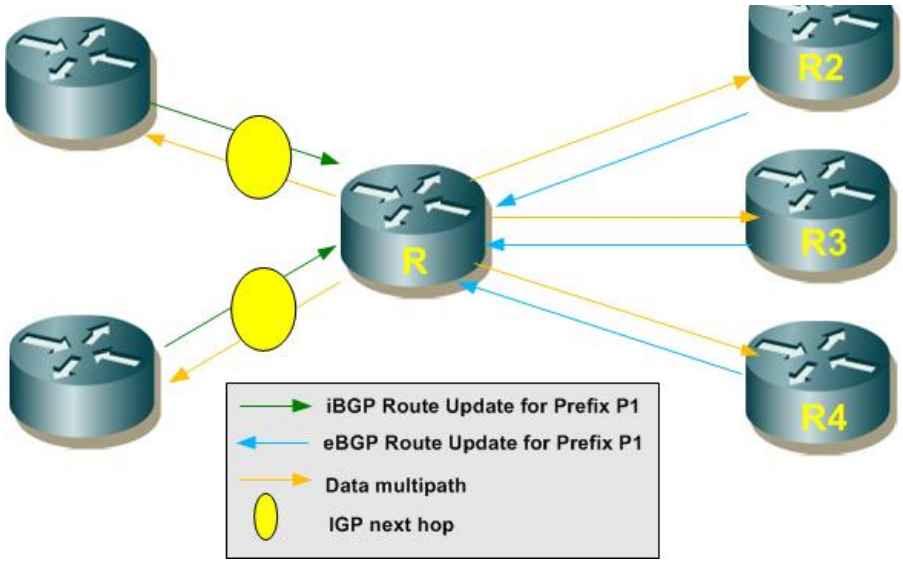



**Figure 17-1: Configuration of Multipath Router for iBGP and eBGP Load Balancing**

NSM provides multipath support for upper-level protocols, such as OSPF and IS-IS. NSM multipath can be configured through a command and compiled with the multipath option.

The BGP multipath feature is used for balancing forwarding loads on eBGP or iBGP routers when the destination route can be reached through multiple “Equal Cost” next hops. Equal cost is determined through the best path selection mechanism and all paths that are of equal cost with the BGP best paths are considered as multipath candidates. In order to be candidates for multipath, paths to the same destination should have the following characteristics equal to the best-path characteristics:

- Weight
- Local preference
- AS-PATH length
- Origin
- MED
- Neighboring AS or sub-AS or AS-PATH

Choose one of the following options at compilation time:

- Sorted order of next-hops in multipath installation (default)
- First Come First Serve (FCFS) multipath installation

# BGP Multipath Architecture

Figure 17-2 illustrates the basic building blocks of the architectural design for BGP multipath.

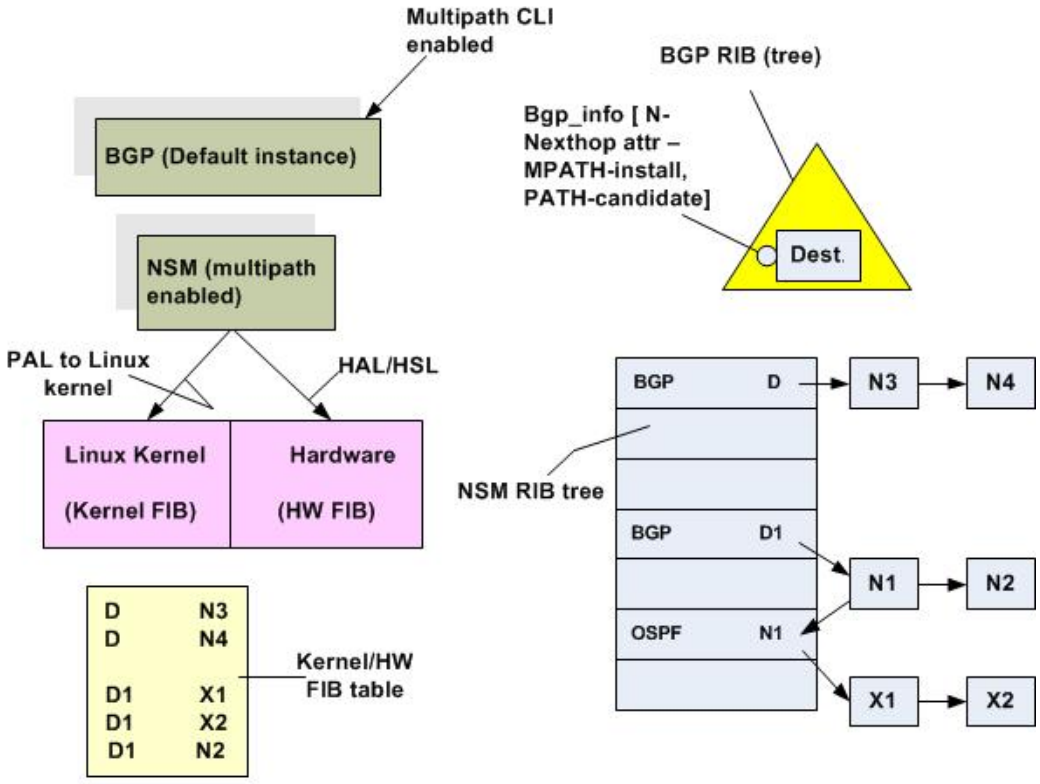



**Figure 17-2: BGP Multipath Architecture Block Diagram**

## BGP Module

The BGP module handles the following functions:

- BGP configuration for multipath
- Keeping track of BGP attribute changes
- Rule changing and updating ECMP multipath flags
- Updating NSM with multipath next-hops

## NSM Module

The NSM module handles the following tasks:

- Maximum-path value configuration
- Updating the IGP next-hops corresponding to a BGP recursive nexthop when the nexthop goes up and down
- Keeping track of multiple IGP next-hops for a recursive BGP nexthop in RIB table

# CHAPTER 18 BGP Data Structures

This chapter describes the data structures that are used in the Border Gateway Protocol (BGP) functions.

The following subsections describe the BGP data structures.

## attr

This structure is located in bgpd/bgp_attr.h.

```
/* BGP attribute structure. */
struct attr
{
  /* Reference count of this attribute. */
  u_int32_t refcnt;

  /* Flag of attribute is set or not. */
  u_int32_t flag;

  /* Partial Flag of optional transitive attribute. */
  u_int8_t partial_flag;
#define BGP_ATTR_AGGREGATOR_PARTIAL  0x80   /* Aggregator attribute is partial. */
#define BGP_ATTR_COMMUNITY_PARTIAL   0x40   /* Community attribute is partial. */
#define BGP_ATTR_ECOMMUNITY_PARTIAL  0x20   /* Extended-Community attribute is partial.
*/

#define BGP_ATTR_AS4_AGGREGATOR_PARTIAL 0x10 /* AS4 Aggregator attribute is partial. */
  /* Attributes. */
  u_int8_t origin;
  u_int8_t distance;
  u_int8_t mp_nexthop_len;

  struct pal_in4_addr nexthop;
  u_int32_t nsm_metric;
  u_int32_t med;
  u_int32_t local_pref;
#ifndef HAVE_EXT_CAP_ASN
  as_t aggregator_as;
#else
  u_int16_t aggregator_as;
  as_t aggregator_as4;
#endif /* HAVE_EXT_CAP_ASN */
  u_int8_t pad2 [2];
  struct pal_in4_addr aggregator_addr;
  /* u_int32_t dpa; */
  u_int32_t weight;
  struct pal_in4_addr originator_id;
```

```
  struct cluster_list *cluster;

#ifdef HAVE_IPV6
  struct pal_in6_addr mp_nexthop_global;
  struct pal_in6_addr mp_nexthop_local;
#endif /* HAVE_IPV6 */
  struct pal_in4_addr mp_nexthop_global_in;

  /* AS Path structure */
#ifndef HAVE_EXT_CAP_ASN
  struct aspath *aspath;
#else
  struct aspath *aspath;
  struct as4path *as4path;
  struct as4path *aspath4B;
#endif /* HAVE_EXT_CAP_ASN */

  /* Community structure */
  struct community *community;

  /* Extended Communities attribute. */
  struct ecommunity *ecommunity;

  /* Unknown transitive attribute. */
  struct transit *transit;

  /* BGP RFD Config Parameter -
   * Used only as a pass-through parameter for 'route_map_apply'
   */
  struct bgp_rfd_cb_cfg_param *rfd_cb_cfg;
};
```

# bgp

This structure is located in bgpd/bgpd.h.

| Member | Description |
|---|---|
| owning_bvr | Pointer to owning BGP virtual router structure |
| owning_ivrf | Pointer to owning Library VRF structure |
| bvrf | Pointer to associated BGP VRF structure |
| as | BGP Instance AS Number |
| pad1 | Padding |
| name | BGP instance name |
| peer_self | Pointer to self peer |

| Member | Description |
|---|---|
| peer_list | Pointer to BGP peers list |
| group_list | Pointer to BGP peer group list |
| router_id | BGP router identifier |
| cluster_id | BGP route reflector cluster ID |
| confed_id | BGP confederation information |
| pad2 | BGP confederation information |
| confed_peers | BGP confederation information |
| confed_peers_cnt | BGP confederation information |
| bgp_cflags | BGP Configuration Flags (all AFs) |
| bgp_sflags | GP Instance Status Flags (all AFs; internal events) |
| bgp_af_cflags [BAAI_MAX][BSAI_MAX] | Note: BGP instance per AF configuration flags |
| bgp_af_sflags [BAAI_MAX][BSAI_MAX] | BGP Instance per AF status flags (internal events) |
| rfd_cfg [BAAI_MAX][BSAI_MAX] | Pointer to BGP route flap dampening control block |
| route [BAAI_MAX][BSAI_MAX] | Pointer to Static route configuration |
| aggregate [BAAI_MAX][BSAI_MAX] | Pointer to aggregate address configuration |
| rib [BAAI_MAX][BSAI_MAX] | Pointer to BGP routing information base |
| redist [BAAI_MAX][IPI_ROUTE_MAX | BGP redistribute configuration |
| rmap [BAAI_MAX][IPI_ROUTE_MAX] | BGP redistribute route-map |
| table_version [BAAI_MAX][BSAI_MAX] | BGP table version |
| distance_ebgp [BAAI_MAX][BSAI_MAX] | BGP distance configuration, default value 0 |
| distance_ibgp [BAAI_MAX][BSAI_MAX] | BGP distance configuration, default value 0 |
| distance_local [BAAI_MAX][BSAI_MAX] | BGP distance configuration, default valuE 0 |
| distance_table | Pointer to BGP distance table |
| default_local_pref | BGP default local preference |
| default_holdtime | BGP default timer |

| Member | Description |
|---|---|
| default_keepalive | BGP default timer |
| peer_index [BAAI_MAX][BSAI_MAX] | Peer Index vector |
| listen_sock_lnode | Pointer to BGP Server (Listen) socket threads list |
| rib_scan_interval | BGP RIB scan interval |
| network_scan_interval | BGP Network scan interval |
| selrt_count[BAAI_MAX] | BGP total count of selected routes per AFI |
| nhop_count[BAAI_MAX] | AFI based Nexthop count for all BGP selected routes |
| nhop_count[BAAI_MAX] | `AFI based Nexthop count for all BGP selected routes` |
| nht_params | Pointer to NHT related parameters |
| rib_scan_tab | Pointer to BGP RIB Scan current-table |
| rib_scan_rn | Pointer to BGP RIB Scan current node |
| t_rib_scan | Pointer to BGP RIB scan thread |
| rdhash_tab | Pointer to BGP RD Hash Table per view indexed by RD |
| rd_list | Pointer to BGP RD List per view searched by RD |
| t_network_scan | Pointer to BGP network scan thread |
| *nh_tab[BAAI_MAX] | Pointer to BGP nexthop tree having nexthops of selected BGP routes. |
| nh_tab[BAAI_MAX] | Pointer to BGP nexthop tree having nexthops of selected BGP routes |
| bnc_ipv4 | Pointer to BGP nexthop cache for IPV4 |
| cache4_1 | Pointer to BGP nexthop cache for IPV4 |
| cache4_2 | Pointer to BGP nexthop cache for IPV4 |
| cache4_nh_tmp | Pointer to Temporary IPV4 nexthop cache used by NHT |
| neighbors_converged | Pointer to BGP convergence |
| conv_complete | Pointer to BGP convergence |
| bnc_ipv6 | Pointer to BGP Nexthop Cache for IPV6 |
| cache6_1 | Pointer to BGP Nexthop Cache for IPV6 |
| cache6_2 | Pointer to BGP Nexthop Cache for IPV6 |
| bnc_ipv6 | Pointer to BGP Nexthop Cache for IPV6 */ |
| cache6_1 | Pointer to BGP Nexthop Cache for IPV6 */ |
| cache6_2 | Pointer to BGP Nexthop Cache for IPV6 */ |

| Member | Description |
|---|---|
| cache6_nh_tmp | Pointer to temporary IPV6 nexthop cache used by NHT |
| t_announce_defer | Pointer to BGP GRST defer route selection thread |
| peer_list_eor [BAAI_MAX][BSAI_MAX] | Pointer to BGP GRST list of peers pending EOR |
| maxpath_ebgp | ECMP multipath |
| maxpath_ibgp | ECMP multipath |
| cfg_maxpath_ebgp | ECMP multipath |
| cfg_maxpath_ibgp | ECMP multipath |
| aslocal_count | AS local count, default is 1 |

```
/* BGP Instance structure */
struct bgp
{
  /* Owning BGP VR structure */
  struct bgp_vr *owning_bvr;

  /* Owning Library VRF structure */
  struct ipi_vrf *owning_ivrf;

#ifdef HAVE_VRF
  /* Associated BGP VRF structure */
  struct bgp_vrf *bvrf;
#endif /* HAVE_VRF */

  /* BGP Instance AS Number */
  as_t as;
  u_int8_t pad1 [2];

  /* BGP Instance Name */
  u_int8_t *name;

  /* Self Peer */
  struct bgp_peer *peer_self;

  /* BGP Peers List */
  struct list *peer_list;

  /* BGP Peer-Group List */
  struct list *group_list;

  /* BGP router identifier.  */
  struct pal_in4_addr router_id;

  /* BGP route reflector cluster ID */
```

```
  struct pal_in4_addr cluster_id;

  /* BGP confederation information.  */
  as_t confed_id;
  u_int8_t pad2 [2];
  as_t *confed_peers;
  u_int16_t confed_peers_cnt;

  /* BGP Configuration Flags (all AFs) */
  u_int32_t bgp_cflags;
#define BGP_CFLAG_ROUTER_ID             (1 << 0)
#define BGP_CFLAG_CLUSTER_ID            (1 << 1)
#define BGP_CFLAG_CLUSTER_ID_DIGIT      (1 << 2)
#define BGP_CFLAG_CONFEDERATION         (1 << 3)
#define BGP_CFLAG_DEFAULT_LOCAL_PREF    (1 << 4)
#define BGP_CFLAG_DEFAULT_TIMER         (1 << 5)
#define BGP_CFLAG_MPLS_RESOLUTION       (1 << 6)
#define BGP_CFLAG_ALWAYS_COMPARE_MED    (1 << 7)
#define BGP_CFLAG_DETERMINISTIC_MED     (1 << 8)
#define BGP_CFLAG_MED_MISSING_AS_WORST  (1 << 9)
#define BGP_CFLAG_MED_CONFED            (1 << 10)
#define BGP_CFLAG_NO_DEFAULT_IPV4       (1 << 11)
#define BGP_CFLAG_NO_CLIENT_TO_CLIENT   (1 << 12)
#define BGP_CFLAG_ENFORCE_FIRST_AS      (1 << 13)
#define BGP_CFLAG_COMPARE_ROUTER_ID     (1 << 14)
#define BGP_CFLAG_ASPATH_IGNORE         (1 << 15)
#define BGP_CFLAG_NO_FAST_EXT_FAILOVER  (1 << 16)
#define BGP_CFLAG_NO_INBOUND_RT_FILTER  (1 << 17)
#define BGP_CFLAG_LOG_NEIGHBOR_CHANGES  (1 << 18)
#define BGP_CFLAG_COMPARE_CONFED_ASPATH (1 << 19)
#define BGP_CFLAG_MED_REMOVE_RCVD       (1 << 20)
#define BGP_CFLAG_MED_REMOVE_SEND       (1 << 21)
#define BGP_CFLAG_ROUTER_DELETE_IN_PROGRESS (1 << 22)
#define BGP_CFLAG_ECMP_ENABLE(1 << 23)
#define BGP_CFLAG_PREFER_OLD_ROUTE(1 << 24)
#define BGP_CFLAG_DONT_COMP_ORIG_ID(1 << 25)
#define BGP_CFLAG_DISALLOW_INFINITE_HOLD_TIME (1 << 26)


  /* BGP Instance Status Flags (all AFs) - For internal events */
  u_int32_t bgp_sflags;
#define BGP_SFLAG_RIB_SCAN_FRESH        (1 << 0)
#define BGP_SFLAG_GRST_SUPPORT          (1 << 1)
#define BGP_SFLAG_GRST_RESTART_ON       (1 << 2)
#define BGP_SFLAG_GRST_GRRESET_SUPPORT  (1 << 3)
#define BGP_SFLAG_GRST_GRRESET          (1 << 4)
#define BGP_SFLAG_NH_CHANGE_RECV        (1 << 5)

  /* BGP Instance per AF Configuration Flags */
  u_int32_t bgp_af_cflags [BAAI_MAX][BSAI_MAX];
```

```
#define BGP_AF_CFLAG_SYNCHRONIZATION          (1 << 0)
#define BGP_AF_CFLAG_NETWORK_SYNC             (1 << 1)
#define BGP_AF_CFLAG_AUTO_SUMMARY             (1 << 2)

  /* BGP Instance per AF Status Flags - For internal events */
  u_int32_t bgp_af_sflags [BAAI_MAX][BSAI_MAX];
#define BGP_AF_SFLAG_TABLE_ANNOUNCED                (1 << 0)
#define BGP_AF_SFLAG_GRST_DEFER_ANNOUNCEMENT        (1 << 1)
#define BGP_AF_SFLAG_GRST_FIB_PRESERVED             (1 << 2)

  /* BGP Route Flap Dampening Control Block */
  struct bgp_rfd_cfg *rfd_cfg [BAAI_MAX][BSAI_MAX];

  /* Static route configuration */
  struct bgp_ptree *route [BAAI_MAX][BSAI_MAX];

  /* Aggregate address configuration */
  struct bgp_ptree *aggregate [BAAI_MAX][BSAI_MAX];

  /* BGP routing information base */
  struct bgp_ptree *rib [BAAI_MAX][BSAI_MAX];

  /* BGP redistribute configuration */
  u_int8_t redist [BAAI_MAX][IPI_ROUTE_MAX];

  /* BGP redistribute route-map */
  struct bgp_rmap rmap [BAAI_MAX][IPI_ROUTE_MAX];

  /* BGP table version */
  u_int32_t table_version [BAAI_MAX][BSAI_MAX];

  /* BGP distance configuration, and the default values are 0 */
  u_int8_t distance_ebgp [BAAI_MAX][BSAI_MAX];
  u_int8_t distance_ibgp [BAAI_MAX][BSAI_MAX];
  u_int8_t distance_local [BAAI_MAX][BSAI_MAX];

  /* BGP distance table */
  struct bgp_ptree *distance_table;

  /* BGP default local-preference */
  u_int32_t default_local_pref;

  /* BGP default timer */
  u_int16_t default_holdtime;
  u_int16_t default_keepalive;

  /* Peer Index vector */
  vector peer_index [BAAI_MAX][BSAI_MAX];

  /* BGP Server (Listen) Socket Threads List */
```

```
  struct bgp_listen_sock_lnode *listen_sock_lnode;

  /* BGP RIB Scan Interval */
  u_int16_t rib_scan_interval;

  /* BGP Network Scan Interval */
  u_int16_t network_scan_interval;

  /* BGP total count of selected routes per AFI */
  u_int32_t selrt_count[BAAI_MAX];

  /* AFI based Nexthop count for all BGP selected routes. */
  u_int32_t nhop_count[BAAI_MAX];

  /*
   * NHT related parameters
  */
  struct bgp_nht_params *nht_params;

  /* BGP RIB Scan current-table */
  struct bgp_ptree *rib_scan_tab;

  /* BGP RIB Scan current-node */
  struct bgp_node *rib_scan_rn;

  /* BGP RIB Scan Thread */
  struct thread *t_rib_scan;

#ifdef HAVE_VRF
  /* BGP RD Hash Table per view indexed by RD */
  struct hash *rdhash_tab;

  /* BGP RD List per view searched by RD */
  struct list *rd_list;
#endif /*HAVE_VRF*/

  /* BGP Network Scan Thread */
  struct thread *t_network_scan;

  /* BGP nexthop tree having nexthops of
   * selected BGP routes.
  */
  struct bgp_ptree *nh_tab[BAAI_MAX];

  /* BGP Nexthop Cache for IPV4 */
  struct bgp_ptree *bnc_ipv4;
  struct bgp_ptree *cache4_1;
  struct bgp_ptree *cache4_2;
  /* Temporary IPV4 nexthop cache used by NHT */
  struct bgp_ptree *cache4_nh_tmp;
```

```
  /* BGP convergence */
  u_int16_t neighbors_converged;
  u_int16_t conv_complete;

#ifdef HAVE_IPV6
  /* BGP Nexthop Cache for IPV6 */
  struct bgp_ptree *bnc_ipv6;
  struct bgp_ptree *cache6_1;
  struct bgp_ptree *cache6_2;
  /* Temporary IPV6 nexthop cache used by NHT */
  struct bgp_ptree *cache6_nh_tmp;
#endif /* HAVE_IPV6 */

#ifdef HAVE_RESTART
  /* BGP GRST Defer Route Selection Thread */
  struct thread *t_announce_defer;

  /* BGP GRST List of Peers pending EOR */
  struct list *peer_list_eor [BAAI_MAX][BSAI_MAX];
#endif /* HAVE_RESTART */

#define BGP_MAXPATH_SUPPORTED64
#define BGP_DEFAULT_MAXPATH_ECMP 1
  /* ECMP MULTIPATH */
  u_int16_t maxpath_ebgp;
  u_int16_t maxpath_ibgp;
  u_int16_t cfg_maxpath_ebgp;
  u_int16_t cfg_maxpath_ibgp;
#define BGP_LOCAL_AS_COUNT_MAX  64
  /* as-local-count: default is 1 */
  u_int16_t aslocal_count;
};
```

## bgp_peer

This structure is located in bgpd/bgpd.h.

```
/* BGP neighbor structure. */
struct bgp_peer
{
  /* BGP structure.  */
  struct bgp *bgp;
  /* master bgp is used in multi-instance allow same peer
   * this instance on which the timer are running */
  struct bgp * master_bgp;

  /* BGP peer group.  */
  struct bgp_peer_group *group;
  u_int8_t af_group [BAAI_MAX][BSAI_MAX];
```

```
  /* Peer's remote AS number. */
  as_t as;

  /* Peer's local AS number. */
  as_t local_as;

  /* Remote router ID. */
  struct pal_in4_addr remote_id;

  /* Local router ID. */
  struct pal_in4_addr local_id;

  /* BGP Peer Notify Data */
  struct bgp_peer_notify_info *notify_info;

  /* BGP list for Route-Server */
  struct list *peer_bgp_node_list;

  /* BGP node for Route-Server in the context of view */
  struct peer_bgp_node *pbgp_node_inctx;

  /* BGP Peer Incoming Peer Connections list */
  struct list *clones_list;

  /* BGP Peer Owning Real-Peer (valid if Clone) */
  struct bgp_peer *real_peer;

  /* BGP Peer Stream Socket CB */
  struct stream_sock_cb *sock_cb;
;
  /* BGP Peer TCP connection TTL */
  u_int8_t ttl;

  /* BGP Peer is on same shared network */
  u_int8_t shared_network;

  /* BGP Peer Port Number */
  u_int16_t sock_port;

  /* Peer information */
  u_int8_t *desc;               /* Description of the peer. */
  u_int8_t *host;               /* Printable address of the peer. */
  union sockunion su;           /* Sockunion address of the peer. */
  pal_time_t uptime;            /* Last Up/Down time */
  pal_time_t last_reset_time;   /* Last Reset time */

  u_int8_t *ifname;             /* bind interface name. */
  u_int8_t *update_if;
  union sockunion *update_source;
```

```
  struct zlog *log;
  u_int8_t version;             /* Peer BGP version. */

  union sockunion *su_local;    /* Sockunion of local address.  */
  union sockunion *su_remote;   /* Sockunion of remote address.  */
  struct bgp_nexthop nexthop;   /* Nexthop */

  /* BGP Peer FSM State */
  u_int32_t bpf_state;

  /* BGP Peer FSM ConnectRetryCounter */
  u_int32_t bpf_conn_retry_count;

  /* BGP peer used in multiple views */
  u_int32_t refcnt;

  /* Peer address family configuration. */
  u_int8_t afc [BAAI_MAX][BSAI_MAX];
  u_int8_t afc_nego [BAAI_MAX][BSAI_MAX];
  u_int8_t afc_adv [BAAI_MAX][BSAI_MAX];
  u_int8_t afc_recv [BAAI_MAX][BSAI_MAX];

  /* Peer index information for each address family.  */
  struct bgp_peer_index index [BAAI_MAX][BSAI_MAX];

  /* Capability Flags.*/
  u_int32_t cap;
#define PEER_CAP_REFRESH_ADV                (1 << 0) /* refresh advertised */
#define PEER_CAP_REFRESH_OLD_RCV            (1 << 1) /* refresh old received */
#define PEER_CAP_REFRESH_NEW_RCV            (1 << 2) /* refresh rfc received */
#define PEER_CAP_DYNAMIC_ADV                (1 << 3) /* dynamic advertised */
#define PEER_CAP_DYNAMIC_RCV                (1 << 4) /* dynamic received */
#define PEER_CAP_NONE_RCV                   (1 << 5) /* No capability received */
#define PEER_CAP_EXTENDED_ASN_ADV           (1 << 6) /* Extended ASN Capability
advertised */
#define PEER_CAP_EXTENDED_ASN_RCV          (1 << 7) /* Extended ASN Capability received
*/
  /* Per AF Capability Flags */
  u_int16_t af_cap [BAAI_MAX][BSAI_MAX];
#define PEER_CAP_ORF_PREFIX_SM_ADV          (1 << 0) /* send-mode advertised */
#define PEER_CAP_ORF_PREFIX_RM_ADV          (1 << 1) /* receive-mode advertised */
#define PEER_CAP_ORF_PREFIX_SM_RCV          (1 << 2) /* send-mode received */
#define PEER_CAP_ORF_PREFIX_RM_RCV          (1 << 3) /* receive-mode received */
#define PEER_CAP_ORF_PREFIX_SM_OLD_RCV      (1 << 4) /* send-mode received */
#define PEER_CAP_ORF_PREFIX_RM_OLD_RCV      (1 << 5) /* receive-mode received */
#define PEER_CAP_GRST_CAPABILITY            (1 << 6) /* graceful-restart */


u_int32_t dyn_cap_flags;
#define PEER_CAP_MP_NEW_DYN_CAP_RCV         (1 << 0) /* Multi- protocol dynamic
capabilty received*/
```

```
#define PEER_CAP_REFRESH_OLD_DYN_CAP_RCV    (1 << 1) /* refresh old dynamic capability
received */
#define PEER_CAP_REFRESH_NEW_DYN_CAP_RCV    (1 << 2) /* refresh rfc dynamic capability
received */
#define PEER_CAP_GRST_DYN_CAP_RCV           (1 << 3) /* graceful-restart dynamic
capability received */

  /* Peer-level (all AFs) configuration flags */
  u_int32_t flags;
#define PEER_FLAG_PASSIVE                   (1 << 0) /* passive mode */
#define PEER_FLAG_SHUTDOWN                  (1 << 1) /* shutdown */
#define PEER_FLAG_DONT_CAPABILITY           (1 << 2) /* dont-capability */
#define PEER_FLAG_OVERRIDE_CAPABILITY       (1 << 3) /* override-capability */
#define PEER_FLAG_STRICT_CAP_MATCH          (1 << 4) /* strict-match */
#define PEER_FLAG_NO_ROUTE_REFRESH_CAP      (1 << 5) /* route-refresh */
#define PEER_FLAG_DYNAMIC_CAPABILITY        (1 << 6) /* dynamic capability */
#define PEER_FLAG_ENFORCE_MULTIHOP          (1 << 7) /* enforce-multihop */
#define PEER_FLAG_COLLIDE_ESTABLISHED       (1 << 8) /* Estab state Collision Detect */
#define PEER_FLAG_NO_IF_BINDING             (1 << 9) /* No Interfaces bound to BGP
Instance */
#define PEER_FLAG_IN_GROUP                  (1 << 10) /* peer-group conf */
#define PEER_FLAG_GROUP_IN_VRF              (1 << 11) /* peer-group is in VRF */
#define PEER_FLAG_6PE_ENABLED               (1 << 12) /* peer is 6pe Enabled */
#define PEER_DISALLOW_INFINITE_HOLD_TIME    (1 << 13) /* peer-disallow-infinite-hold-
time */
#define PEER_FLAG_RECV_EOR_UPDATE           (1 << 14) /* EOR update recieved */
#ifdef HAVE_BFD
#define PEER_FLAG_BFD                       (1 << 15) /* BFD */
#endif /* HAVE_BFD */
#define PEER_FLAG_LOCAL_AS                  (1 << 16) /* Local-AS override */
#define PEER_FLAG_VERSION_CHECK             (1 << 17) /* Bgp Version check */
#define PEER_FLAG_L2VPN_NO_SOCK_PURGE       (1 << 18) /* Bgp l2vpn no sock purge check
*/

  /* Per AF Configuration Flags */
  u_int32_t af_flags [BAAI_MAX][BSAI_MAX];
#define PEER_FLAG_SEND_COMMUNITY            (1 << 0) /* send-community */
#define PEER_FLAG_SEND_EXT_COMMUNITY        (1 << 1) /* send-community ext. */
#define PEER_FLAG_NEXTHOP_SELF              (1 << 2) /* next-hop-self */
#define PEER_FLAG_REFLECTOR_CLIENT          (1 << 3) /* reflector-client */
#define PEER_FLAG_RSERVER_CLIENT            (1 << 4) /* route-server-client */
#define PEER_FLAG_SOFT_RECONFIG             (1 << 5) /* soft-reconfiguration */
#define PEER_FLAG_AS_PATH_UNCHANGED         (1 << 6) /* transparent-as */
#define PEER_FLAG_NEXTHOP_UNCHANGED         (1 << 7) /* transparent-next-hop */
#define PEER_FLAG_MED_UNCHANGED             (1 << 8) /* transparent-med */
#define PEER_FLAG_DEFAULT_ORIGINATE         (1 << 9) /* default-originate */
#define PEER_FLAG_REMOVE_PRIVATE_AS         (1 << 10) /* remove-private-as */
#define PEER_FLAG_ALLOWAS_IN                (1 << 11) /* set allowas-in */
#define PEER_FLAG_ORF_PREFIX_SM             (1 << 12) /* orf capability send-mode */
#define PEER_FLAG_ORF_PREFIX_RM             (1 << 13) /* orf capability receive-mode */
#define PEER_FLAG_MAX_PREFIX_WARNING        (1 << 14) /* maximum prefix warning-only */
#define PEER_FLAG_AS_OVERRIDE               (1 << 15) /* AS override  */
```

```
#define PEER_FLAG_SITE_ORIGIN               (1 << 16) /* set site-origin */
#define PEER_FLAG_GRST_CAPABILITY           (1 << 17) /* graceful-restart */
#define PEER_FLAG_EBGP_VPN_ALLOW            (1 << 18) /* Allow EBGP VPN */

  /* Peer-level (all AFs) Status Flags */
  u_int32_t sflags;
#define PEER_STATUS_PREFIX_OVERFLOW         (1 << 0) /* prefix-overflow */
#define PEER_STATUS_CAPABILITY_OPEN         (1 << 1) /* capability open send */
#define PEER_STATUS_SOFT_RESET_IN           (1 << 2) /* Soft-reset In */
#define PEER_STATUS_SOFT_RESET_OUT          (1 << 3) /* Soft-reset Out */
#define PEER_STATUS_CAP_ROUTE_REFRESH       (1 << 4) /* Capability RR modification */
#define PEER_STATUS_CONV_FOR_IGP            (1 << 5) /* Conv. status for IGP */
#define PEER_STATUS_G_SHUT                  (1 << 6) /* node g-shut state*/

  /* Per AF Status Flags */
  u_int16_t af_sflags [BAAI_MAX][BSAI_MAX];
#define PEER_STATUS_ORF_PREFIX_SEND         (1 << 0) /* prefix-list send peer */
#define PEER_STATUS_ORF_WAIT_REFRESH        (1 << 1) /* wait refresh received peer */
#define PEER_STATUS_ORF_NOT_WAIT_REFRESH    (1 << 2) /* not waiting refresh */
#define PEER_STATUS_AF_DEFAULT_ORIGINATE    (1 << 3) /* default-originate peer */
#define PEER_STATUS_AF_SOFT_RESET_IN        (1 << 4) /* Soft-reset In */
#define PEER_STATUS_AF_SOFT_RESET_OUT       (1 << 5) /* Soft-reset Out */
#define PEER_STATUS_AF_GRST_CAPABILITY      (1 << 6) /* GRST Config change */
#define PEER_STATUS_AF_ROUTE_REFRESH_SEND   (1 << 7) /* Send Route-Refresh */
#define PEER_STATUS_AF_ROUTE_REFRESH_RCVD   (1 << 8) /* Received Route-Refresh */
#define PEER_STATUS_AF_ASORIG_ROUTE_ADV     (1 << 9) /* Advt. AS-Origin Routes */

  /* Default attribute value for the peer. */
  u_int32_t config;
#define PEER_CONFIG_TIMER                   (1 << 1) /* keepalive & holdtime */
#define PEER_CONFIG_CONNECT                 (1 << 2) /* connect */
#define PEER_CONFIG_ASORIG                  (1 << 3) /* route advertise */
#define PEER_CONFIG_ROUTEADV                (1 << 4) /* route advertise */
#define PEER_CONFIG_PASSWORD                (1 << 5) /* MD5 password.  */
#define PEER_CONFIG_GRST_RESTART            (1 << 6) /* GRST Restart */
#ifdef HAVE_BFD
#define PEER_CONFIG_BFD                     (1 << 7) /* BFD */
#define PEER_CONFIG_BFD_MH                  (1 << 8) /* BFD multihop */
#endif /* HAVE_BFD */
#define PEER_CONFIG_ROUTEADV_IMMEDIATE      (1 << 9) /* route advertise
                                                        immediate. */
  u_int32_t weight [BAAI_MAX][BSAI_MAX];
  u_int32_t holdtime;
  u_int32_t keepalive;
  u_int32_t connect;
  u_int32_t asorig;
  u_int32_t routeadv;

  /* BGP Peer MD5 Auth setting */
  u_int8_t password_type;
```

```
u_int8_t *password;

/* BGP Peer FSM Timer values */
u_int32_t v_auto_start;
u_int32_t v_connect;
u_int32_t v_holdtime;
u_int32_t v_keepalive;
u_int32_t v_asorig;
u_int32_t v_routeadv;
u_int32_t v_gshut_time;

/* BGP Peer FSM Timer Threads */
struct thread *t_auto_start;
struct thread *t_connect;
struct thread *t_holdtime;
struct thread *t_keepalive;
struct thread *t_asorig;
struct thread *t_routeadv;
struct thread *t_gshut_timer;

/* Statistics fields */
u_int32_t open_in;            /* Open message input count */
u_int32_t open_out;           /* Open message output count */
u_int32_t update_in;          /* Update message input count */
u_int32_t update_out;         /* Update message ouput count */
u_int32_t keepalive_in;       /* Keepalive input count */
u_int32_t keepalive_out;      /* Keepalive output count */
u_int32_t notify_in;          /* Notify input count */
u_int32_t notify_out;         /* Notify output count */
u_int32_t refresh_in;         /* Route Refresh input count */
u_int32_t refresh_out;        /* Route Refresh output count */
u_int32_t dynamic_cap_in;     /* Dynamic Capability input count.  */
u_int32_t dynamic_cap_out;    /* Dynamic Capability output count.  */

/* BGP state count */
u_int32_t established;        /* Established */
u_int32_t dropped;            /* Dropped */

/* BGP peer ID in the peer group */
s_int32_t peer_id;

/* BGP Peer Advertisement lists for non-AS-Origin routes */
struct bgp_peer_adv_list *adv_list [BAAI_MAX][BSAI_MAX];

/* BGP Peer Advertisement lists for AS-Origination routes */
struct bgp_peer_adv_list *asorig_adv_list [BAAI_MAX][BSAI_MAX];

/* BGP Peer Advertisement lists for non-AS-Origin routes */
struct bgp_peer_adv_list *adv_list_new [BAAI_MAX][BSAI_MAX];
```

```
/* BGP Peer Advertisement lists for AS-Origination routes */
struct bgp_peer_adv_list *asorig_adv_list_new [BAAI_MAX][BSAI_MAX];


/* Current advertisement in the same attribute */
struct bgp_advertsise *curr_adv [BAAI_MAX][BSAI_MAX];
struct bgp_advertsise *curr_asorig [BAAI_MAX][BSAI_MAX];

/* Update message received time */
pal_time_t update_time;

/* BGP Peer Previous Advertisement Time */
pal_time_t advtime;

/* Send prefix count. */
u_int32_t scount [BAAI_MAX][BSAI_MAX];

/* BGP Peer Advertisement Attribute Hash Table */
struct hash *baa_hash [BAAI_MAX][BSAI_MAX];

/* Filter structure. */
struct bgp_filter filter [BAAI_MAX][BSAI_MAX];

/* ORF Prefix-list */
struct prefix_list *orf_plist [BAAI_MAX][BSAI_MAX];

/* Prefix count. */
u_int32_t pcount [BAAI_MAX][BSAI_MAX];

/* Table version.  */
u_int32_t table_version [BAAI_MAX][BSAI_MAX];

/* BGP Peer Maximum Acceptable Prefix Limit */
u_int32_t pmax [BAAI_MAX][BSAI_MAX];

/* BGP Peer Maximum Acceptable Prefix Warning Threshold */
u_int32_t threshold [BAAI_MAX][BSAI_MAX];

/* allowas-in. */
u_int8_t allowas_in [BAAI_MAX][BSAI_MAX];

/* FIFO of Decoded UPDATE Messges Info. */
struct fifo bdui_fifo;

/* FIFO of Incoming Connection Requests */
struct fifo bicr_fifo;

/* default-originate route-map.  */
struct bgp_rmap default_rmap [BAAI_MAX][BSAI_MAX];
```

```
#ifdef HAVE_VRF
  /* site-of-origin */
  struct bgp_rd site_origin [BAAI_MAX][BSAI_MAX];
#endif /* HAVE_VRF */

  /* to check whether a default-originated route
     is already sent to this peer.
     if sent this variable will be PAL_TRUE
     else it will be PAL_FALSE.
  */
  bool_t def_orig_route_sent;

#ifdef HAVE_RESTART
  /* BGP GRST Configured Restart Timer */
  u_int32_t grst_restart;

  /* BGP GRST Timer values */
  u_int32_t v_grst_preserve;
  u_int32_t v_grst_restart;

  /* BGP GRST Timer Threads */
  struct thread *t_grst_preserve;
  struct thread *t_grst_restart;

  /* Peer-level GRST Status Flags */
  u_int16_t gstatus;
#define PEER_GSTATUS_RESTART_ON         (1 << 0)  /* GRST Peer re-established,
convergence underway */
#define PEER_GSTATUS_ACCEPT_GRST        (1 << 1)  /* GRST Cap Negotiate */
#define PEER_GSTATUS_PRESERVE_ON        (1 << 2)  /* GRST Peer is down and stale routes
are being preserved */

  /* Per-AF GRST Status Flags */
  u_int8_t gstatus_af [BAAI_MAX][BSAI_MAX];
#define PEER_GSTATUS_AF_GRST_ADV        (1 << 0)  /* GRST Cap Advertised */
#define PEER_GSTATUS_AF_GRST_RCV        (1 << 1)  /* GRST Cap Received */
#define PEER_GSTATUS_AF_GRST_NEGO       (1 << 2)  /* GRST Cap Negotiated */
#define PEER_GSTATUS_AF_FWD_PRESERVE    (1 << 3)  /* Restarting Side Preserved Stale
Paths in FIB */
#define PEER_GSTATUS_AF_RIB_PRESERVE    (1 << 4)  /* Receiving Side Preserved Stale
Paths in RIB */

  /* FIFO of Deferred Decoded UPDATE Messges Info. */
  struct fifo bdui_fifo_deferred;
#endif /* HAVE_RESTART */
};
```

# bgp_peer_group

This structure is located in bgpd/bgpd.h.

| Member | Description |
|---|---|
| name | Pointer to the peer group name |
| bgp | Pointer to the owning BGP structure |
| peer_list | Pointer to the peer group client list |
| bgp_peer | Pointer to the peer group configuration |

```
/* BGP Peer-Group structure */
struct bgp_peer_group
{
  /* Peer-Group Name */
  u_int8_t *name;

  /* Owining BGP structure  */
  struct bgp *bgp;

  /* Peer-group client list. */
  struct list *peer_list;

  /* Peer-group config */
  struct bgp_peer *conf;
}
```

# CHAPTER 19 BGP Command API

This chapter describes the Border Gateway Protocol (BGP) command API functions.

## API Functions

Following is a summary of the BGP functions. Details are provided in the following subsections.

| Function | Description |
|---|---|
| bgp_auto_summary_update | Advertises or withdraws summarized routes with BGP peers |
| bgp_cluster_id_set | Sets the ID of the BGP route reflector cluster |
| bgp_cluster_id_unset | Unsets the ID of the BGP route reflector cluster |
| bgp_conf_ext_asn_cap | Handles the dynamic change of the extended ASN capability option |
| bgp_confederation_id_set | Sets the ID of the BGP Confederation |
| bgp_confederation_id_unset | Unsets the ID of the BGP confederation |
| bgp_confederation_peers_add | Adds a new BGP Confederation peer's AS number |
| bgp_confederation_peers_remove | Deletes the BGP Confederation peer's AS number |
| bgp_default_local_preference_set | Sets the default local preference |
| bgp_default_local_preference_unset | Unsets the default local preference |
| bgp_delete | Deletes the specified BGP instance |
| bgp_distance_config_set | Sets the external, internal, and local distances to the corresponding eBGP, iBGP, and local distance values in the BGP structure |
| bgp_distance_config_unset | Unsets the previously configured eBGP, iBGP, and local distance values |
| bgp_extcommunity_list_set | Adds a new entry to community-list, |
| bgp_get | Returns the pointer to the specified BGP instance |
| bgp_graceful_reset_cap | Implements the graceful reset capability flag setting |
| bgp_network_sync_set | Ensures the specified static network prefix has IGP reachability |
| bgp_network_sync_unset | (to be filled in when information is available) |
| bgp_option_check | Checks the set state of the specified option |
| bgp_option_set | Sets the BGP option |
| bgp_option_unset | Unsets the specified BGP option |
| bgp_peer_clear | Invokes the BGP stop event for a peer |

| Function | Description |
|---|---|
| bgp_peer_delete | Deletes the specified peer |
| bgp_peer_g_shut_time_set | Sets the shutdown timer value for a BGP peer that is enabled with the graceful shutdown capability |
| bgp_peer_g_shut_time_unset | Unsets a configured shutdown timer value for a BGP peer and returns the value to the default value |
| bgp_peer_group_bind | Binds a peer to specified peer-group |
| bgp_peer_group_delete | Deletes the specified peer-group |
| bgp_peer_group_remote_as | Specifies a peer-group's AS number |
| bgp_peer_group_remote_as_delete | Removes peer-group's AS configuration |
| bgp_peer_group_unbind | Unbinds a peer from a specified peer-group |
| bgp_peer_remote_as | Creates a new peer |
| bgp_router_id_unset | Unsets the BGP router ID |
| bgp_scan_time_set | Sets the next-hop scan time |
| bgp_scan_time_unset | Resets the next-hop scan time to the default value |
| bgp_session_g_shut | Initiates the graceful shutdown of a BGP peer session |
| bgp_session_g_no_shut | Brings up a BGP peer session after a graceful shutdown of the session |
| bgp_synchronization_set | Enables IGP synchronization of iBGP routes |
| bgp_synchronization_unset | Disables IGP synchronization of iBGP routes |
| bgp_timers_set | Sets the BGP configuration timer |
| bgp_timers_unset | Resets the BGP configuration timer |
| peer_activate | Activates specified peer's Address Family configuration |
| peer_advertise_interval_set | Sets a peer's advertise interval timer |
| peer_advertise_interval_unset | Sets back the peer's advertise interval timer to the default value |
| peer_af_flag_check | Checks whether or not the peer's address family configuration flag is set |
| peer_af_flag_set | Set the peer's address family only configuration flag |
| peer_af_flag_unset | Unsets peer's address family only configuration flag |
| peer_allow_ebgp_vpn | Allows a VPN connection with an eBGP peer |
| peer_disallow_ebgp_vpn | Disallows a VPN connection with an eBGP peer |
| peer_allowas_in_set | Enables the AS path loop check for MPLS VPN/BGP environment |
| peer_allowas_in_unset | Unsets allowas-in configuration |

| Function | Description |
|---|---|
| peer_aslist_set | Sets the peer's filter-list configuration |
| peer_aslist_unset | Unsets peer's filter-list configuration |
| peer_clear_soft | Performs soft reconfiguration in/out, and Outbound Routing Filter (ORF) prefix list refresh |
| peer_deactivate | Deactivates the specified peer's Address Family configuration |
| peer_default_originate_set | Sets the peer's default-originate configuration |
| peer_default_originate_unset | Unsets the peer's default-originate configuration |
| peer_description_set | Sets the peer's description string |
| peer_description_unset | Clears the peer's description string |
| peer_disallow_hold_timer_set | Disallows peer’s configuration of infinite hold-time |
| peer_disallow_hold_timer_unset | Allows a peer’s configuration of infinite hold-time |
| peer_distribute_set | Sets peer's distribute-list filter |
| peer_distribute_unset | Unsets peer's distribute-list filter |
| peer_ebgp_multihop_set | Sets the eBGP multihop configuration |
| peer_ebgp_multihop_set | Resets the eBGP multihop configuration of the peer |
| peer_flag_check | Checks whether or not peer configuration flag is set |
| peer_flag_set | Sets the peer configuration flag |
| peer_flag_unset | Unsets the peer configuration flag |
| peer_interface_set | Sets the peer's interface for IPv6 link-local address configuration |
| peer_interface_unset | Unsets the peer's interface for IPv6 link-local address configuration |
| peer_maximum_prefix_set | Sets maximum prefix configuration |
| peer_maximum_prefix_unset | Unsets the maximum prefix configuration |
| peer_port_set | Sets the BGP port number |
| peer_port_unset | Unsets BGP port number |
| peer_prefix_list_set | Sets peer's prefix-list filter |
| peer_prefix_list_unset | Unsets peer's route-map configuration |
| peer_route_map_set | Sets peer's route-map configuration |
| peer_route_map_unset | Unsets peer's route-map configuration |
| peer_timers_connect_set | Sets peer's connect timer value |
| peer_timers_connect_unset | Unsets the peer’s connect timer value |

| Function | Description |
|---|---|
| peer_timers_set | Sets the peer's timer value |
| peer_timers_unset | Unsets the peer's timer value |
| peer_unsuppress_map_set | Specifies peer's unsuppress-map configuration |
| peer_unsuppress_map_unset | Unsets the unsuppress-map configuration of the peer |
| peer_update_source_addr_set | Sets peer's update source IP address |
| peer_update_source_if_set | Sets peer's update source interface name |
| peer_update_source_unset | Unsets all update source configuration |
| peer_weight_set | Sets the BGP peer's weight value |
| peer_weight_unset | Unsets the weight configuration |

## bgp_auto_summary_update

This function advertises or withdraws summarized routes with BGP peers.

This function is called by the following command:

```
auto-summary
```

### Syntax

```
s_int32_t
bgp_auto_summary_update (struct bgp *bgp,afi_t afi,afi_t safi,
                         bool_t auto_summary_set);
```

### Input Parameters

`bgp` BGP instance

`afi` Address family

`safi` Sub-Address family

`auto_summary_set`
Summary set of type

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when API call is successful (0)

BGP_API_INVALID_ROUTE_NODE when route node is invalid (-61)

BGP_API_SET_ERR_AUTO_SUMMARY_ENABLED when auto summary is enabled (-62)

BGP_API_SET_ERR_AUTO_SUMMARY_DISABLED when auto summary is disabled (-63)

## bgp_cluster_id_set

This function sets the ID of the BGP route reflector cluster.

This function is called by the following command:

```
bgp cluster-id A.B.C.D
```

### Syntax

```
int
bgp_cluster_id_set (struct bgp *bgp, struct pal_in4_addr *cluster_id);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |
| `cluster_id` | BGP route reflector cluster ID |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_cluster_id_unset

This function unsets the ID of the BGP route reflector cluster.

This function is called by the following command:

```
no bgp cluster-id
```

### Syntax

```
int
bgp_cluster_id_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_conf_ext_asn_cap

This function handles the dynamic change of the extended ASN capability option.

This function is called by the following command:

```
bgp extended-asn-cap
```

### Syntax

```
s_int32_t
bgp_conf_ext_asn_cap (struct bgp *bgp, u_int32_t flag, bool_t set);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `flag` | Extended ASN flag |
| `set` | True/False |

### Output Parameters

None

### Return Value

BGP_API_SET_ERR_INVALID_BGP when the BGP instance is invalid (-5)

BGP_API_SET_ERR_ALREADY_EXTASNCAP when extended ASN capability is already enabled (-53)

BGP_API_SET_ERR_NO_EXTASNCAP when extended ASN capability is not enabled, and an attempt is made to configure BGP with 4-octet non-mappable ASN (-54)

BGP_API_SET_ERR_NONMAPPABLE when BGP is configured with 4-octet non-mappable ASN, and an attempt is made to disable extended ASN capability (-56)

BGP_API_SET_ERR_INVALID_REMOTEASN when Remote AS Number is invalid (-64)

## bgp_confederation_id_set

This function sets the ID of the BGP Confederation.

This function is called by the following command:

```
bgp confederation identifier
```

### Syntax

```
int
bgp_confederation_id_set (struct bgp *bgp, as_t as);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |
| `as` | Confederation ID <1–65535> |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

GP_API_SET_ERR_INVALID_AS when the AS value is Invalid (-4)

## bgp_confederation_id_unset

This function unsets the ID of the BGP confederation.

This function is called by the following command:

```
no bgp confederation identifier
```

### Syntax

```
int
bgp_confederation_id_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_confederation_peers_add

This function adds a new BGP Confederation peer's AS number.

This function is called by the following command:

```
bgp confederation peers
```

### Syntax

```
int
bgp_confederation_peers_add (struct bgp *bgp, as_t as);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |
| `as` | Confederation ID |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_GET_SUCCESS 0 when bgp_confederation_peers_check is successful (0)

BGP_API_SET_ERR_INVALID_BGP when BGP instance is equal to zero (-5)

BGP_API_SET_ERR_INVALID_AS when local member-AS not allowed in confed peer list (-4)

## bgp_confederation_peers_remove

This function deletes the BGP Confederation peer's AS number.

This function is called by the following command:

```
no bgp confederation peers
```

### Syntax

```
int
bgp_confederation_peers_remove (struct bgp *bgp, as_t as);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |
| `as` | Confederation ID |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful. (0)

## bgp_default_local_preference_set

This function sets the default local preference.

This function is called by the following command:

```
bgp default local-preference
```

### Syntax

```
int
bgp_default_local_preference_set (struct bgp *bgp, u_int32_t local_pref);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |
| `local_pref` | BGP default local preference < 0–4294967295>; default value is 100 |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_default_local_preference_unset

This function resets the default local preference

This function is called by the following command:

```
bgp default local-preference
```

### Syntax

```
int
bgp_default_local_preference_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | A pointer to BGP instance |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_delete

This function deletes the specified BGP instance.

This function is called by the following command:

```
no router bgp
```

### Syntax

```
s int32
t bgp_delete (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `*bgp` | BGP instance pointer to be deleted |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_distance_config_set

This function sets the external, internal, and local distances to the corresponding eBGP, iBGP, and local distance values in the BGP structure.

This function is called by the following command:

```
distance bgp
```

### Syntax

```
void
bgp_distance_config_set (struct cli *cli, char *distance_ebgp,
char *distance_ibgp, char *distance_local);
```

### Input Parameters

| | |
|---|---|
| `cli` | Pointer to the CLI structure |
| `distance_ebgp` | eBGP distance <1–255> |
| `distance_ibgp` | iBGP distance <1–255> |
| `distance_local` | Local distance <1–255> |

### Output Parameters

None

### Return Value

None

## bgp_distance_config_unset

This function unsets the previously configured eBGP, iBGP, and local distance values.

This function is called by the following command:

```
no distance bgp
```

### Syntax

```
int
bgp_distance_config_unset(struct cli *cli, u_int32_t distance_ebgp,
u_int32_t distance_ibgp, u_int32_t distance_local);
```

### Input Parameters

| | |
|---|---|
| `cli` | Pointer to the CLI structure |
| `distance_ebgp` | eBGP distance <1–255> |
| `distance_ibgp` | iBGP distance <1–255> |
| `distance_local` | Local distance <1–255> |

### Output Parameters

None

### Return Value

CLI_SUCCESS when the CLI is successful (0)

CLI_ERROR when the unsetting is done with values other than the previously configured values

## bgp_extcommunity_list_set

This function adds a new entry to community-list, If the list does not exist, it creates a new list, then adds the entry.

This function is called by the following command:

```
ip extcommunity-list
```

### Syntax

```
int
bgp_extcommunity_list_set (char *name, char *str, int type,
int direct, int style);
```

### Input Parameters

| | |
|---|---|
| `name` | The extended community-list name |
| `str` | The extended community string. |
| `type` | community-list name type:<br>(0) COMMUNITY_LIST_STRING<br>(1) COMMUNITY_LIST_NUMBER |
| `direct` | BGP API filter direct:<br>(0) BGP_API_FILTER_DENY<br>(1) BGP_API_FILTER_PERMIT |
| `style` | standard/expanded extended community list:<br>(3) EXTCOMMUNITY_LIST_STANDARD<br>(4) EXTCOMMUNITY_LIST_EXPANDED<br>(5) EXTCOMMUNITY_LIST_AUT0 |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the call is successful (0)

BGP_API_SET_ERROR when community list is empty (-1)

BGP_API_SET_ERR_CLIST_DEFINE_CONFLICT when there is a Community name conflict with previously defined (-39)

BGP_API_SET_ERR_MALFORMED_ARG when community list type is EXTCOMMUNITY_LIST_STANDARD (-33)

## bgp_get

This function returns the pointer to the specified BGP instance. If no pointer is returned, it tries to create a new one.

This function is called by the following command:

```
router bgp
```

### Syntax

```
s_int32_t
bgp_get (struct bgp **bgp_val, as_t *as, u_int8_t *name);
```

### Input Parameters

| | |
|---|---|
| `bgp_val` | Returns BGP instance pointer |
| `as` | AS number |
| `name` | BGP instance name: optional, usually NULL |

### Output Parameters

| | |
|---|---|
| `bgp_val` | Returned BGP instance pointer |

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERROR when the function call failed (-1)

BGP_API_SET_ERR_MULTIPLE_INSTANCE_NOT_SET when name is specified but BGP multiple instance is not enabled (-15)

BGP_API_SET_ERR_AS_MISMATCH when BGP is already running under different AS number1 (-16)

BGP_API_SET_ERR_INSTANCE_MISMATCH when AS number is different with existing configuration (-29)

BGP_API_NHT_NOT_ENABLED_SET_ERR when Nexthop tracking is not enabled (-68)

## bgp_graceful_reset_cap

This function implements the graceful reset capability flag setting. This capability flag, BGP_SFLAG_GRST_GRRESET_SUPPORT, determines whether BGP supports graceful reset.

This function is called by the following command:

```
bgp graceful-restart graceful-reset
```

### Syntax

```
s_int32_t
bgp_graceful_reset_cap (struct bgp *bgp, bool_t set);
```

### Input Parameters

| | |
|---|---|
| `bgp` | master BGP instance structure |
| `set` | flag to indicate whether to set or unset the graceful reset capability |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERROR when the BGP instance does not exist (-1)

BGP_API_SET_ERR_NO_GRST_SUPPORT when graceful restart is not enabled (-27)

## bgp_network_sync_set

This function ensures the specified static network prefix has IGP reachability in the NSM RIB before being introduced into the BGP RIB. The network to be advertised by the BGP routing process is specified as follows:

- A unicast network address without a mask is accepted if it falls into the natural boundary of its class.
- A class-boundary mask is derived if the address matches its natural class-boundary.

This function is called by the following command:

```
network synchronization
```

### Syntax

```
s_int32_t
bgp_network_sync_set (struct bgp *bgp, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `afi` | Address family |
| `safi` | Sub Address family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the call is successful (0)

## bgp_network_sync_unset

This function resets the configured network synchronization.

This function is called by the following command:

```
no network synchronization
```

### Syntax

```
s_int32_t
bgp_network_sync_unset (struct bgp *bgp, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `afi` | Address family |
| `safi` | Sub Address family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the call is successful (0)

API Error Codes

## bgp_option_check

This function checks the set state of the specified option.

### Syntax

```
int
bgp_option_check (u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `flag` | BGP option:<br>BGP_OPT_NO_FIB—do not install BGP route into FIB<br>BGP_OPT_MULTIPLE_INSTANCE—multiple instance<br>BGP_OPT_CONFIG_STANDARD—industry standard oriented configuration<br>BGP_OPT_RFC1771_PATH_SELECT—RFC1771 style path selection<br>BGP_OPT_RFC1771_STRICT—strictly follow RFC1771 description<br>BGP_OPT_AGGREGATE_NEXTHOP_CHECK—aggregate route only when next hop is same<br>BGP_OPT_ANVL_DAMPENING_CONFIG—ANVL style dampening config parse<br>BGP_OPT_EXTENDED_ASN_CAP—extended ASN capability |

### Output Parameters

None

### Return Value

1 when the option is set

0 when the option is not set

## bgp_option_set

This function sets the BGP option. The BGP option is a system-wide pre-configurable setting, and is usually not accessible to the end user.

This function is called by the following command:

```
bgp multiple-instance
```

### Syntax

```
int
bgp_option_set (int flag);
```

### Input Parameters

| | |
|---|---|
| `flag` | BGP options:<br>BGP_OPT_NO_FIB—do not install BGP route into FIB<br>BGP_OPT_MULTIPLE_INSTANCE—multiple instance<br>BGP_OPT_CONFIG_STANDARD—industry-standard oriented configuration<br>BGP_OPT_RFC1771_PATH_SELECT—RFC1771 style path selection<br>BGP_OPT_RFC1771_STRICT—strictly follow RFC1771 description<br>BGP_OPT_AGGREGATE_NEXTHOP_CHECK—aggregate route only when next hop is same<br>BGP_OPT_ANVL_DAMPENING_CONFIG—ANVL style dampening config parse<br>BGP_OPT_EXTENDED_ASN_CAP—extended ASN capability |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_MULT_INST_DEL_CONFIG when trying to delete multi instance configurations (-43)

BGP_API_SET_ERR_ADJ_OUT_DYNAMIC -78 when trying to change this option while BGP instance is running (-78)

## bgp_option_unset

This function unsets the specified BGP option.

This function is called by the following command:

```
no bgp multiple-instance
```

### Syntax

```
int
bgp_option_unset (u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `flag` | BGP options:<br>BGP_OPT_NO_FIB—do not install BGP route into FIB<br>BGP_OPT_MULTIPLE_INSTANCE—multiple instance<br>BGP_OPT_CONFIG_STANDARD—industry-standard oriented configuration<br>BGP_OPT_RFC1771_PATH_SELECT—RFC1771 style path selection<br>BGP_OPT_RFC1771_STRICT—strictly follow RFC1771 description<br>BGP_OPT_AGGREGATE_NEXTHOP_CHECK—aggregate route only when next hop is same<br>BGP_OPT_ANVL_DAMPENING_CONFIG—ANVL style dampening config parse<br>BGP_OPT_EXTENDED_ASN_CAP—extended ASN capability |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_ADJ_OUT_DYNAMIC -78; Can not change this option while BGP instance running (-78)

BGP_API_SET_ERR_MULT_INST_DEL_CONFIG -43; when trying to delete multi instance configurations (-43)

## bgp_peer_clear

This function invokes the BGP stop event for a peer. The peer's connect will be shut down, and all of the route will be cleared.

This function is called by the following command:

```
clear ip bgp
```

### Syntax

```
s_int32_t
bgp_peer_clear (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_peer_delete

This function deletes the specified peer.

### Syntax

```
s_int32_t
bgp_peer_delete (struct bgp_peer *peer)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful

## bgp_peer_g_shut_time_set

This function sets the shutdown timer value for a BGP peer that is enabled with the graceful shutdown capability. After the timer expires, the sessions of a BGP peer are shut down. For details about the graceful shutdown capability, see Graceful Shutdown.

This function is called by the following command:

```
neighbor A.B.C.D g-shut-timer <10-65535>
```

### Syntax

```
s_int32_t
bgp_peer_g_shut_time_set (struct bgp_peer *peer, u_int32_t shut_time);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `shut_time` | The maximum time in seconds specified for the value of the graceful shutdown timer |

### Output Parameters

None

### Return Value

BGP_API_SET_ERR_NO_CAP_CMD when the peer is not enabled with the graceful shutdown capability

BGP_API_SET_SUCCESS when the call is successful

## bgp_peer_g_shut_time_unset

This function unsets a configured shutdown timer value for a BGP neighbor or peer and returns the value to the default value (60 seconds).

This function is called by the following command:

```
no neighbor A.B.C.D g-shut-timer <10-65535>
```

### Syntax

```
s_int32_t
bgp_peer_g_shut_time_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_ERR_NO_CAP_CMD when the peer is not enabled with the graceful shutdown capability.

BGP_API_SET_SUCCESS when the function is successful

## bgp_peer_group_bind

This function binds a peer to specified peer-group. When a peer does not exist, it creates a new peer.

This function is called by the following command:

```
peer-group <group name>
```

### Syntax

```
s_int32_t
bgp_peer_group_bind (struct bgp *bgp, union sockunion *su,
struct bgp_peer_group *group, afi_t afi, safi_t safi, as_t *as)
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `su` | Peer's IP address with unison sockunion format |
| `group` | BGP peer-group |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

| | |
|---|---|
| `as` | AS number when error occur |

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP__UNCONFIGURED when the peer-group is not activated for the specified I and SI (-10)

BGP_API_SET_ERR_PEER_GROUP_NO_REMOTE_AS when the peer-group does not have AS configuration (-11)

BGP_API_SET_ERR_PEER_GROUP_CANT_CHANGE when the peer already belongs to different peer-group (-12)

BGP_API_SET_ERR_PEER_GROUP_MISMATCH when the peer already belongs to different peer-group for a different Address Family configuration (-13)

BGP_API_SET_ERR_PEER_GROUP_PEER_TYPE_DIFFERENT when a try occurs to create an iBGP peer inside an eBGP peer-group or vice versa (-14)

## bgp_peer_group_delete

This function deletes the specified peer-group.

This function is called by the following command:

```
peer-group <name>
```

### Syntax

```
s_int32_t
bgp_peer_group_delete (struct bgp_peer_group *group);
```

### Input Parameters

| | |
|---|---|
| `group` | BGP peer-group |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_peer_group_remote_as

This function specifies a peer-group's AS number. When a peer-group member already has AS configuration, this call forces to change the configuration to a new one.

This function is called by the following command:

```
remote-as
```

### Syntax

```
s_int32_t
peer_group_remote_as (struct bgp *bgp, u_int8_t *group_name, as_t *as);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `group_name` | Peer-group name |
| `as` | AS number |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERROR when the Peer group is empty (-1)

BGP_API_SET_ERR_AS_MISMATCH when the AS number does not match (-16)

## bgp_peer_group_remote_as_delete

This function removes peer-group's AS configuration.

This function is called by the following command:

```
remote-as
```

### Syntax

```
s_int32_t
bgp_peer_group_remote_as_delete (struct bgp_peer_group *group);
```

### Input Parameters

| | |
|---|---|
| `group` | Peer group name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_peer_group_unbind

This function unbinds a peer from a specified peer-group.

This function is called by the following command:

```
no peer-group <WORD>
```

### Syntax

```
s_int32_t
bgp_peer_group_unbind (struct bgp *bgp, struct bgp_peer *peer,
struct bgp_peer_group *group, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `peer` | BGP peer |
| `group` | BGP peer-group |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_MISMATCH when a specified peer does not belongs to the peer-group (-13)

## bgp_peer_remote_as

This function creates a new peer.

This function is called by the following command:

```
remote-as
```

### Syntax

```
s_int32_t
bgp_remote_as (struct bgp *, union sockunion *su, as_t *as, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `su` | Peer's IP address with union sockunion format |
| `as` | AS number of the peer |
| `afi` | Address family identifier (I_IP, I_IP6) |
| `safi` | Subsequent address family identifier (SI_UNICAST, SI_MULTICAST) |

### Output Parameters

| | |
|---|---|
| `as` | When an error occurs, this AS value is used to return the AS value to the caller |

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERROR -1; when Peer is enabled in the same view (-1)

BGP_API_SET_ERR_PEER_GROUP_MEMBER when this peer is a peer-group member, and the peer-group already has AS configuration (-6)

BGP_API_SET_ERR_PEER_GROUP_PEER_TYPE_DIFFERENT when try to create iBGP peer inside eBGP peer-group or vice versa (-14)

BGP_API_SET_ERR_AS_MISMATCH when AS no mismatch (-16)

BGP_API_SET_ERR_AS_MISMATCH when there is a mismatch in the AS (-16)

BGP_API_SET_ERR_AS_MISMATCH when the AS number of peer and instance should not match with master instance AS number (-16)

BGP_API_SET_ERR__UNCONFIGURED when address family not unconfigured (-25)

BGP_API_SET_ERR_PEER_CONFIG_IN_ANOTHER_INST when peer is configured in another instance (-45)

## bgp_router_id_unset

This function unsets the BGP router ID. Afterward, a new router ID is picked up from the interface address.

This function is called by the following command:

```
no bgp router-id A.B.C.D
```

### Syntax

```
int
router_id_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## bgp_scan_time_set

This function sets the next-hop scan time.

This function is called by the following command:

```
bgp scan-time
```

### Syntax

```
s_int32_t
bgp_scan_time_set (struct bgp *bgp, u_int32_t scan_interval);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance in VR case, otherwise, use NULL. |
| `scan_interval` | scan interval in seconds <0–60>; default value is 60 |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when BGP scan time is set successfully (0)

BGP_API_SET_ERR_INVALID_VALUE when scan time interval is more than maximum scan time interval (-2)

BGP_API_SET_ERR_NHT_DISABLE_RIB_SCAN when disabling RIB-SCAN is not allowed: NHT is enabled (-74)

## bgp_scan_time_unset

This function resets the next-hop scan time to the default value: 60 seconds.

This function is called by the following command:

```
no bgp scan-time
```

### Syntax

```
s_int32_t
bgp_scan_time_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance in VR case; otherwise, use NULL |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when API is successful (0)

BGP_API_SET_ERROR when modify scan interval fails (-1)

## bgp_session_g_shut

This function initiates the graceful shutdown of a BGP peer session. The session is brought down after the shutdown timer expires.

This function is called by the following command:

```
neighbor  <neighbor address>  g-shut
```

For details about the graceful shutdown feature, see Graceful Shutdown.

### Syntax

```
s_int32_t
bgp_session_g_shut (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address family |
| `safi` | Sub Address family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful

BGP_API_SET_ERR_PEER_IBGP when the graceful shutdown is not allowed on an IBGP peer

BGP_API_SET_ERR_NO_CAP_CMD when the peer is not enabled with the graceful shutdown capability

BGP_API_SET_ERR_PEER_G_SHUT when the router cannot perform graceful shutdown because the peer is not UNICAST

BGP_API_SET_ERR_ALREADY_G_SHUT when the router/neighbor is already in the graceful shutdown mode

## bgp_session_g_no_shut

This function brings up a BGP peer session after a graceful shutdown of the session.

This function is called by the following command:

```
no neighbor  <neighbor address>  g-shut
```

For details about the graceful shutdown feature, see Graceful Shutdown.

### Syntax

```
s_int32_t
bgp_session_no_g_shut (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful

BGP_API_SET_ERR_NO_G_SHUT when the router/neighbor is not the graceful shutdown mode

BGP_API_SET_ERR_UNDER_G_SHUT when the BGP session is still up; wait until the session goes down

## bgp_synchronization_set

This function enables IGP synchronization of iBGP routes.

This function is called by the following command:

```
synchronization
```

### Syntax

```
s_int32_t ret
bgp_synchronization_set (struct bgp *bgp, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `afi` | Address family |
| `safi` | Sub Address family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when API is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when BGP instance is empty (-2)

## bgp_synchronization_unset

This function disables IGP synchronization of iBGP routes.

This function is called by the following command:

```
no synchronization
```

### Syntax

```
s_int32_t
bgp_synchronization_unset (struct bgp *bgp, afi_t afi, safi_t safi)
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `afi` | Address family |
| `safi` | Sub Address family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when API is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when BGP instance is empty (-2)

## bgp_timers_set

This function sets the BGP configuration timer.

This function is called by the following command:

```
timers bgp
```

### Syntax

```
int
bgp_timers_set (struct bgp *bgp, u_int32_t keepalive, u_int32_t holdtime);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |
| `keepalive` | BGP keepalive time in seconds, 0–65535 |
| `holdtime` | BGP holdtime in seconds, 0–65535 |

### Output Parameters

`None`

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE -2; when hold time or keep alive time exceeds maximum value: 65535 (-2)

BGP_API_SET_ERR_INFINITE_HOLD_TIME_VALUE when hold time is infinite value (-57)

BGP_API_SET_WARN_HOLD_AND_KEEPALIVE_INVALID when the configured holdtime is not equal to 0 and the hold time less than the 3 times of keepalive time (-58)

BGP_API_SET_ERR_INVALID_HOLD_TIME when Hold time value is not equal to 0 and less than 3 (-59)

BGP_API_SET_ERR_HOLD_LESS_EQUAL_KEEPALIVE when hold time is not equal to 0 and less than or equal to keep alive time (-79)

## bgp_timers_unset

This function resets the BGP configuration timer.

This function is called by the following command:

```
no timers bgp
```

### Syntax

```
int
bgp_timers_unset (struct bgp *bgp);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP instance |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## peer_activate

This function activates specified peer's Address Family configuration.

This function is called by the following command:

```
acttivate
```

### Syntax

```
int
peer_activate (struct bgp *bgp, struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `bgp` | BGP Instance |
| `peer` | BGP peer |
| `afi` | Address Family |
| `safi` | Subsequent Address Family |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful: peer address is configured correctly (0)

BGP_API_SET_ERROR when peer group is empty (-1)

BGP_API_SET_ERR_PEER_GROUP__INVALID when Invalid combination of address families for this peer-group (-41)

BGP_API_SET_ERR_UNSUP_VPNVF_CONF when it can't activate VPNV4 family for EBGP peer (-47)

## peer_advertise_interval_set

This function sets a peer's advertise interval timer. The default value of advertise interval is 5 seconds for iBGP peers and 30 seconds for eBGP peers.

This function is called by the following command:

```
advertisement-interval
```

### Syntax

```
int
peer_advertise_interval_set (struct bgp_peer *peer, u_int32_t routeadv);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `routeadv` | minimum route advertisement interval value |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when routeadv value is greater than 600 (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when peer is already configured for some address family (-18)

BGP_API_SET_ERR_ALREADY_SET when the configurations are already set (-77)

## peer_advertise_interval_unset

This function sets back the peer's advertise interval timer to the default value. The default value of advertise interval is 5 seconds for iBGP peers and 30 seconds for eBGP peers.

This function is called by the following command:

```
advertisement-interval
```

### Syntax

```
int
peer_advertise_interval_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when peer is already configured for some address family (-18)

## peer_af_flag_check

This function checks whether the address family configuration flag of the peer is set.

### Syntax

```
s_int32_t
peer_af_flag_check (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer. |
| `afi` | Address Family Identifier. |
| `safi` | Subsequent Address Family Identifier |
| `flag` | Peer address family only configuration flag:<br>PEER_FLAG_SEND_COMMUNITY—unset neighbor send-community flag<br>PEER_FLAG_SEND_EXT_COMMUNITY—unset neighbor send-community extended flag<br>PEER_FLAG_NEXTHOP_SELF—unset neighbor next-hop-self flag<br>PEER_FLAG_REFLECTOR_CLIENT—unset neighbor route-reflector-client flag<br>PEER_FLAG_RSERVER_CLIENT—unset neighbor route-server-client flag<br>PEER_FLAG_SOFT_RECONFIG—unset neighbor soft-reconfiguration inbound flag<br>PEER_FLAG_AS_PATH_UNCHANGED—unset neighbor attribute-unchanged as-path flag |

PEER_FLAG_NEXTHOP_UNCHANGED—unset neighbor attribute-unchanged next-hop flag

PEER_FLAG_MED_UNCHANGED—unset neighbor attribute-unchanged med flag

PEER_FLAG_REMOVE_PRIVATE_AS—unset neighbor remove-private-AS flag

PEER_FLAG_AS_OVERRIDE—unset neighbor as-override flag

PEER_FLAG_GRST_CAPABILITY—unset neighbor capability graceful-restart flag

### Output Parameters

None

### Return Value

1 when flag is set

0 when flag is not set

## peer_af_flag_set

This function set the peer's address family only configuration flag.

### Syntax

```
s_int32_t
peer_af_flag_set (struct bgp_peer *peer, afi_t afi, safi_t safi, u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `flag` | Peer address family only configuration flag: |
| | PEER_FLAG_SEND_COMMUNITY—set neighbor send-community flag |
| | PEER_FLAG_SEND_EXT_COMMUNITY—set neighbor send-community extended flag |
| | PEER_FLAG_NEXTHOP_SELF—set neighbor next-hop-self flag |
| | PEER_FLAG_REFLECTOR_CLIENT—set neighbor route-reflector-client flag |
| | PEER_FLAG_RSERVER_CLIENT—set neighbor route-server-client flag |
| | PEER_FLAG_SOFT_RECONFIG—set neighbor soft-reconfiguration inbound flag |
| | PEER_FLAG_AS_PATH_UNCHANGED—set neighbor attribute-unchanged as-path flag |
| | PEER_FLAG_NEXTHOP_UNCHANGED—set neighbor attribute-unchanged next-hop flag |
| | PEER_FLAG_MED_UNCHANGED—set neighbor attribute-unchanged med flag |
| | PEER_FLAG_REMOVE_PRIVATE_AS—set neighbor remove-private-AS flag |
| | PEER_FLAG_AS_OVERRIDE—set neighbor as-override flag |
| | PEER_FLAG_GRST_CAPABILITY—set neighbor capability graceful-restart flag |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FLAG when specified flag is invalid (-3)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not activated on specified I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when specified flag cannot be used for peer- group member peer (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when this peer is a peer-group member; change the peer-group configuration (-19)

BGP_API_SET_ERR_NOT_INTERNAL_PEER when specified flag cannot be used for iBGP peer (-23)

BGP_API_SET_ERR_REMOVE_PRIVATE_AS when user tries to set PEER_FLAG_REMOVE_PRIVATE_AS for iBGP peer. (-24)

## peer_af_flag_unset

This function unsets peer's address family only configuration flag.

### Syntax

```
s_int32_t
(peer_af_flag_unset (struct bgp_peer *peer, afi_t afi, safi_t safi, u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `flag` | Peer address family only configuration flag: |
| | PEER_FLAG_SEND_COMMUNITY—unset neighbor send-community flag |
| | PEER_FLAG_SEND_EXT_COMMUNITY—unset neighbor send-community extended flag |
| | PEER_FLAG_NEXTHOP_SELF—unset neighbor next-hop-self flag |
| | PEER_FLAG_REFLECTOR_CLIENT—unset neighbor route-reflector-client flag |
| | PEER_FLAG_RSERVER_CLIENT—unset neighbor route-server-client flag |
| | PEER_FLAG_SOFT_RECONFIG—unset neighbor soft-reconfiguration inbound flag |
| | PEER_FLAG_AS_PATH_UNCHANGED—unset neighbor attribute-unchanged as-path flag |
| | PEER_FLAG_NEXTHOP_UNCHANGED—unset neighbor attribute-unchanged next-hop flag |
| | PEER_FLAG_MED_UNCHANGED—unset neighbor attribute-unchanged med flag |
| | PEER_FLAG_REMOVE_PRIVATE_AS—unset neighbor remove-private-AS flag |
| | PEER_FLAG_AS_OVERRIDE—unset neighbor as-override flag |
| | PEER_FLAG_GRST_CAPABILITY—unset neighbor capability graceful-restart flag |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FLAG when specified flag is invalid (-3)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not activated on specified I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when specified flag cannot be used for peer- group member peer (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when a user tries to unset the flag for the peer which belongs to peer-group, and the peer-group has the flag (-19)

BGP_API_SET_ERR_NOT_INTERNAL_PEER when specified flag cannot be used for iBGP peer (-23)

BGP_API_SET_ERR_REMOVE_PRIVATE_AS when a user tries to unset PEER_FLAG_REMOVE_PRIVATE_AS for iBGP peer (-24)

## peer_allow_ebgp_vpn

This function allows a VPN connection with an eBGP peer.

This function is called by the following command:

```
allow-ebgp-vpn
```

### Syntax

```
int
peer_allow_ebgp_vpn (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safI` | subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP__INVALID when peer group address family (-41)

BGP_API_SET_ERR_UNSUP_VPNVF_CONF when trying to activate VPNV4 family for EBGP peer (-47)

## peer_disallow_ebgp_vpn

This function disallows a VPN connection with an eBGP peer. This is the default behavior for BGP VPN connection.

This function is called by the following command:

```
allow-ebgp-vpn
```

### Syntax

```
int
peer_disallow_ebgp_vpn (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_MEMBER when peer group is empty (-6)

## peer_allowas_in_set

This function enables the AS path loop check for MPLS VPN/BGP environment.

This function is called by the following command:

```
allowas-in
```

### Syntax

```
int
peer_allowas_in_set (struct bgp_peer *peer, afi_t afi, safi_t safi, int allow_num);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `allow_num` | Allow AS number <1 –10> |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when allow_num is invalid (-2)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is peer group member (-19)

## peer_allowas_in_unset

This function unsets `allowas-in` configuration.

This function is called by the following command:

```
allowas-in
```

### Syntax

```
int
peer_allowas_in_unset (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER Invalid command for a peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is peer group member (-19)

## peer_aslist_set

This function sets the peer's filter-list configuration. It is AS path access-list filter.

### Syntax

```
int
peer_aslist_set (struct bgp_peer *peer, afi_t afi, safi_t safi, u_int32_t direct,
u_int8_t *name);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be FILTER_IN or FILTER_OUT. |
| `name` | AS path access-list name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

## peer_aslist_unset

This function unsets peer's filter-list configuration.

### Syntax

```
int
peer_aslist_unset (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be FILTER_IN or FILTER_OUT |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT; for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_clear_soft

This function performs soft reconfiguration in/out, and Outbound Routing Filter (ORF) prefix list refresh.

This function is called by the following command:

```
clear ip bgp
```

### Syntax

```
int
peer_clear_soft (struct bgp_peer *peer, afi_t afi, safi_t safi, u_int32_t stype);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `stype` | Clear type values: |
| | BGP_CLEAR_SOFT_OUT—outbound soft clear |
| | BGP_CLEAR_SOFT_IN—inbound soft clear |
| | BGP_CLEAR_SOFT_BOTH—both direction soft clear |
| | BGP_CLEAR_SOFT_IN_ORF_PREFIX—ORF prefix soft update |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when stype is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_SOFT_RECONFIG_UNCONFIGURED when the peer does not send route-refresh capability and soft-reconfiguration inbound is not configured for the peer (-26)

BGP_API_SET_ERR__UNCONFIGURED when address family is not configured (-27)

## peer_deactivate

This function deactivates the specified peer's Address Family configuration.

This function is called by the following command:

```
activate
```

### Syntax

```
void
peer_deactivate (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family |
| `safi` | Subsequent Address Family |

### Output Parameters

None

### Return Value

None

## peer_default_originate_set

This function sets the peer's default-originate configuration.

This function is called by the following command:

```
default-originate
```

### Syntax

```
int
peer_default_originate_set (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int8_t *rmap)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `rmap` | Default-originate route-map configuration |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## peer_default_originate_unset

This function unsets the peer's default-originate configuration.

This function is called by the following command:

```
default-originate
```

### Syntax

```
int
peer_default_originate_unset (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer. |
| `afi` | Address Family Identifier. |
| `safi` | Subsequent Address Family Identifier. |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID when SI is not unicast (-48)

BGP_API_SET_ERR_PEER_INACTIVE -17 when the peer is inactive (-17)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when this peer is a peer-group member; change peer-group configuration (-17)

## peer_description_set

This function sets the peer's description string.

This function is called by the following command:

```
description
```

### Syntax

```
int
peer_description_set (struct bgp_peer *peer, u_int8_t *desc);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `desc` | Peer's description string |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful

## peer_description_unset

This function clears the peer's description string.

This function is called by the following command:

```
description
```

### Syntax

```
int
peer_description_unset (struct bgp_peer *peer);
```

### Input Parameters

`peer` - BGP peer

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## peer_disallow_hold_timer_set

This function disallows peer’s configuration of infinite hold-time.

This function is called by the following command:

```
disallow-infinite-holdtime
```

### Syntax

```
int
peer_disallow_hold_timer_set (struct bgp_peer *peer);
```

### Input Parameters

`peer` BGP peer

### Output Parameters

None

### Return Value

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is not configured in the group or any of the address family (-18)

## peer_disallow_hold_timer_unset

This function allows a peer's configuration of infinite hold-time.

This function is called by the following command:

```
disallow-infinite-holdtime
```

### Syntax

```
int
peer_disallow_hold_timer_unset (struct bgp_peer *peer)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is not configured in the group or any of the address family (-18)

## peer_distribute_set

This function sets peer's distribute-list filter. This sets access-list filter to the peer.

This function is called by the following command:

```
distribute-list
```

### Syntax

```
int
peer_distribute_set (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct, u_int8_t *name);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be FILTER_IN or FILTER_OUT |
| `name` | Access-list name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when a peer-group member: change peer-group configuration (-19)

BGP_API_SET_ERR_PEER_FILTER_CONFLICT when prefix-list is already configured to the peer (-22)

## peer_distribute_unset

This function unsets peer's distribute-list filter.

This function is called by the following command:

```
distribute-list
```

### Syntax

```
int
peer_distribute_unset (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter: FILTER_IN or FILTER_OUT |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT; peer-group members, outgoing filter is not set for peer-group member (-18)

## peer_ebgp_multihop_set

This function sets the eBGP multihop configuration.

This function is called by the following command:

```
ebgp-multihop
```

### Syntax

```
int
peer_ebgp_multihop_set (struct bgp_peer *peer, u_int8_t ttl)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `ttl` | eBGP multihop TTL value <0–255> |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when peer is a peer-group member: change peer-group configuration (-19)

## peer_ebgp_multihop_set

This function resets the eBGP multihop configuration of the peer.

This function is called by the following command:

```
no ebgp-multihop
```

### Syntax

```
int
peer_ebgp_multihop_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when peer is a peer-group member: change peer-group configuration (-19)

## peer_flag_check

This function checks whether or not peer configuration flag is set.

### Syntax

```
s_int32_t
peer_flag_check (struct bgp_peer *peer, u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `flag` | Peer's configuration flag for all address family |
| | PEER_FLAG_PASSIVE—unset neighbor passive flag |
| | PEER_FLAG_SHUTDOWN—unset neighbor shutdown flag |
| | PEER_FLAG_DONT_CAPABILITY—unset neighbor dont-capability-negotiate flag |
| | PEER_FLAG_OVERRIDE_CAPABILITY—unset neighbor override-capability flag |
| | PEER_FLAG_STRICT_CAP_MATCH—unset neighbor strict-capability-match flag |
| | PEER_FLAG_NO_ROUTE_REFRESH_CAP—unset no neighbor capability route-refresh flag |
| | PEER_FLAG_DYNAMIC_CAPABILITY—unset neighbor capability dynamic flag |
| | PEER_FLAG_ENFORCE_MULTIHOP—unset neighbor enforce-multihop flag<br>PEER_FLAG_COLLIDE_ESTABLISHED—neighbor collide establishment flag |

### Output Parameters

None

### Return Value

1 when flag is set

0 when flag is not set

## peer_flag_set

This function sets the peer configuration flag.

### Syntax

```
s_int32_t
peer_flag_set (struct bgp_peer *peer, u_int32_t flag);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `flag` | Peer's configuration flag for all address family |
| | PEER_FLAG_PASSIVE—set neighbor passive flag |
| | PEER_FLAG_SHUTDOWN—set neighbor shutdown flag |
| | PEER_FLAG_DONT_CAPABILITY—set neighbor dont-capability-negotiate flag |
| | PEER_FLAG_OVERRIDE_CAPABILITY—set neighbor override-capability flag |

PEER_FLAG_STRICT_CAP_MATCH—set neighbor strict-capability-match flag

PEER_FLAG_NO_ROUTE_REFRESH_CAP—set no neighbor capability route-refresh flag

PEER_FLAG_DYNAMIC_CAPABILITY—set neighbor capability dynamic flag

PEER_FLAG_ENFORCE_MULTIHOP—set neighbor enforce-multihop flag

PEER_FLAG_COLLIDE_ESTABLISHED—neighbor collide establishment flag

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when success

BGP_API_SET_ERR_INVALID_FLAG when specified flag is invalid

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when specified flag cannot be used for peer-group member peer

BGP_API_SET_ERR_PEER_FLAG_CONFLICT when user tries to specify conflicting flag PEER_FLAG_STRICT_CAP_MATCH and PEER_FLAG_OVERRIDE_CAPABILITY at the same time

## peer_flag_unset

This function resets the peer configuration flag

### Syntax

```
s_int32_t
peer_flag_unset (struct bgp_peer *peer, u_int32_t flag);
```

### Input Parameters

`peer` BGP peer

`flag` Peer's configuration flag for all address family

PEER_FLAG_PASSIVE—set neighbor passive flag

PEER_FLAG_SHUTDOWN—set neighbor shutdown flag

PEER_FLAG_DONT_CAPABILITY—set neighbor dont-capability-negotiate flag

PEER_FLAG_OVERRIDE_CAPABILITY—set neighbor override-capability flag

PEER_FLAG_STRICT_CAP_MATCH—set neighbor strict-capability-match flag

PEER_FLAG_NO_ROUTE_REFRESH_CAP—set no neighbor capability route-refresh flag

PEER_FLAG_DYNAMIC_CAPABILITY—set neighbor capability dynamic flag

PEER_FLAG_ENFORCE_MULTIHOP—set neighbor enforce-multihop flag

PEER_FLAG_COLLIDE_ESTABLISHED—neighbor collide establishment flag

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FLAG when specified flag is invalid (-3)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when specified flag can't used for peer-group member peer (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when user tries to unset the flag for the peer which belongs to peer-group, and the peer-group has the flag (-19)

BGP_API_SET_ERR_PEER_FLAG_CONFLICT when user tries to specify conflicting flag PEER_FLAG_STRICT_CAP_MATCH and PEER_FLAG_OVERRIDE_CAPABILITY at the same time (-20)

BGP_API_SET_ERR_PEER_GROUP_SHUTDOWN when user tries to no neighbor shutdown for the peer which belongs to peer-group, and the peer-group has neighbor shutdown flag (-21)

BGP_API_SET_ERR_NO_GRST_SUPPORT when BGP graceful-restart not configured (-27)

## peer_interface_set

This function sets the peer's interface for IPv6 link-local address configuration.

This function is called by the following command:

```
interface
```

### Syntax

```
int
peer_interface_set (struct bgp_peer *peer, u_int8_t *ip_str, u_int8_t *str);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `ip_str` | Interface name for IPv6 link-local address configuration |
| `str` | Interface name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when address family is invalid and ip address is invalid (-2)

BGP_API_IP_NOT_IN_SAME_SUBNET when interface IP address and neighbor IP are not in same sub net (-51)

## peer_interface_unset

This function unsets the peer's interface for IPv6 link-local address configuration.

This function is called by the following command:

```
interface
```

### Syntax

```
int
peer_interface_unset (struct bgp_peer *peer, u_int8_t *str);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `str` | Interface name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERROR when peer’s interface name is empty (-1)

BGP_API_INVALID_INTERFACE_NAME when interface name is invalid (-52)

## peer_maximum_prefix_set

This function sets maximum prefix configuration.

This function is called by the following command:

```
maximum-prefix
```

### Syntax

```
int
peer_maximum_prefix_set (struct bgp_peer *peer, afi_t afi, safi_t safi,
                         u_int32_t max, u_int32_t threshold, bool_t warning);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `max` | Maximum number of prefixes acceptable from the neighbor. |
| `threshold` | Threshold number of prefixes from the neighbor. |
| `warning` | If this value is non-zero, it outputs warning instead of shutting down the peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_VALUE when threshold value is invalid (-2)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_maximum_prefix_unset

This function unsets the maximum prefix configuration.

This function is called by the following command:

```
maximum-prefix
```

### Syntax

```
int
peer_maximum_prefix_unset (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

`peer` - BGP peer

`afi` - Address Family Identifier

`safi`—subsequent Address Family Identifier

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_port_set

This function sets the BGP port number.

This function is called by the following command:

```
port
```

### Syntax

```
int
peer_port_set (struct bgp_peer *peer, u_int16_t port);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `port` | BGP port number |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when command for a peer-group member is invalid (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_port_unset

This function unsets BGP port number.

This function is called by the following command:

```
port
```

### Syntax

```
int
peer_port_unset (struct bgp_peer peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_prefix_list_set

This function sets peer's prefix-list filter. This sets prefix-list filter to the peer.

This function is called by the following command:

```
prefix-list
```

### Syntax

```
int
peer_prefix_list_set (struct bgp_peer *peer, afi_t afi, safi_t safi, u_int32_t direct,
u_int8_t *name)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter: |
| | FILTER_IN |
| | FILTER_OUT |
| `name` | Prefix-list name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_FILTER_CONFLICT when distribute-list is already configured to the peer (-22)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_prefix_list_unset

This function unsets peer's prefix-list filter.

This function is called by the following command:

```
prefix-list
```

### Syntax

```
int
peer_prefix_list_unset (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be `FILTER_IN` or `FILTER_OUT`. |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_route_map_set

This function sets peer's route-map configuration.

This function is called by the following command:

```
route-map
```

### Syntax

```
int
peer_route_map_set (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct, u_int8_t *name);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be FILTER_IN or FILTER_OUT |
| `name` | Route-map name |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_route_map_unset

This function unsets peer's route-map configuration.

This function is called by the following command:

```
route-map
```

### Syntax

```
int
peer_route_map_unset ((struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int32_t direct);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `direct` | Direction of the filter. This must be FILTER_IN or FILTER_OUT |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid (-2)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_timers_connect_set

This function sets peer's connect timer value.

This function is called by the following command:

```
connection-retry-time
```

### Syntax

```
int
peer_timers_connect_set (struct bgp_peer *peer, u_int32_t connect);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `connect` | Connect timer value |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when connect timer value is invalid: maximum value is 65536 (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_timers_connect_unset

This function unsets the peer’s connect timer value.

This function is called by the following command:

```
no connection-retry-time
```

### Syntax

```
int
peer_timers_connect_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_timers_set

This function sets the peer's timer value.

This function is called by the following command:

`timersW`

### Syntax

```
int
peer_timers_set (struct bgp_peer *peer, u_int32_t keepalive, u_int32_t holdtime);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `keepalive` | Keepalive value |
| `holdtime` | Holdtime value |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_VALUE when direct value is invalid keepalive time is invalid (maximum 65535); holdtime value is invalid (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

BGP_API_SET_ERR_INFINITE_HOLD_TIME_VALUE when the configured holdtime 0 is not allowed (-57)

BGP_API_SET_WARN_HOLD_AND_KEEPALIVE_INVALID when the configured holdtime and keepalive time are invalid (-58)

BGP_API_SET_ERR_INVALID_HOLD_TIME when holdtime is invalid: minimum configured holdtime is 3 (-59)

## peer_timers_unset

This function unsets the peer's timer value.

This function is called by the following command:

```
timers
```

### Syntax

```
int
peer_timers_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_unsuppress_map_set

This function specifies peer's unsuppress-map configuration.

This function is called by the following command:

```
unsuppress-map
```

### Syntax

```
int
peer_unsuppress_map_set (struct bgp_peer *peer, afi_t afi, safi_t safi,
u_int8_t *name);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |
| `name` | Route-map name for unsuppress-map |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_unsuppress_map_unset

This function unsets the unsuppress-map configuration of the peer.

This function is called by the following command:

```
unsuppress-map
```

### Syntax

```
int
peer_unsuppress_map_unset (struct bgp_peer *peer, afi_t afi, safi_t safi);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_INACTIVE when the peer is not active for the I and SI (-17)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the peer is peer-group member and direct is FILTER_OUT for peer-group member, outgoing filter is not set for peer-group member (-18)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_update_source_addr_set

This function sets peer's update source IP address. This setting overrides peer_update_source_if_set configuration.

This function is called by the following command:

```
update-source
```

### Syntax

```
int
peer_update_source_addr_set (struct bgp_peer *peer, union sockunion *su);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `su` | IP address for update source. |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

## peer_update_source_if_set

This function sets peer's update source interface name.

This function is called by the following command:

```
update-source
```

### Syntax

```
int
peer_update_source_if_set (struct bgp_peer *peer, u_int8_t *ifname);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `ifname` | Interface name string for update source |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_update_source_unset

This function resets all update source configuration.

This function is called by the following command:

```
update-source
```

### Syntax

```
int
peer_update_source_unset (struct bgp_peer *peer);
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_weight_set

This function sets the BGP peer's weight value.

This function is called by the following command:

```
weight
```

### Syntax

```
int
peer_weight_set (struct bgp_peer *peer, u_int16_t weight, afi_t afi, safi_t safi)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `weight` | Weight value |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful (0)

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when the peer is a peer-group member: change peer-group configuration (-19)

## peer_weight_unset

This function unsets the weight configuration.

This function is called by the following command:

```
weight
```

### Syntax

```
int
peer_weight_unset (struct bgp_peer *peer, afi_t afi, safi_t safi)
```

### Input Parameters

| | |
|---|---|
| `peer` | BGP peer |
| `afi` | Address Family Identifier |
| `safi` | Subsequent Address Family Identifier |

### Output Parameters

None

### Return Value

BGP_API_SET_SUCCESS when the function is successful

BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG when there is an error in setting the peer group (-19)

# CHAPTER 20 BGP MIB Support

This chapter describes the BGP API for management information base (MIB) in ZebOS-XP.

## Overview of MIB Implementation

The `BGP4-mib.txt` file contains the MIB definitions for SNMP (based on the definitions in RFC 1657).

## Supported Tables

The following tables are described in this chapter:

- General Variables parameters that apply globally to the Router's BGP Process
- Peer contains one entry per BGP peer, and information about the BGP peer

### General Variables

bgpVersion

| Attribute | Syntax | Access | Function |
|---|---|---|---|
| bgpVersion | Octet-string | Read-only | bgp_get_version |

### Peer

bgpVersion

| Attribute | Syntax | Access | Function |
|---|---|---|---|
| bgpPeerAdminStatus | Integer | Read-write | bgp_set_peer_admin_status |
| bgpPeerLocalAddr | Integer | Read-write | bgp_get_peer_local_addr |
| bgpPeerConnectRetryInterval | Integer32 | Read-write | bgp_set_next_peer_connect_retry_interval |
| bgpPeerHoldTime | Integer32 | Read-only | bgp_get_peer_hold_time |
| bgpPeerHoldTimeConfigured | Integer32 | Read-write | bgp_set_peer_hold_time_configured |

| Attribute | Syntax | Access | Function |
|---|---|---|---|
| bgpPeerKeepAliveConfigured | Integer32 | Read-write | bgp_set_peer_keep_alive_configured |
| bgpPeerMinRouteAdvertisementInterval | Integer32 | Read-write | bgp_set_peer_min_route_advertisement_interval |

# MIB Definitions

## bgp_get_version

This function returns the version of the supported BGP version.

### Syntax

```
int
bgp_get_version (u_int32_t vr_id, int proc_id, u_char *version);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | Process ID |

### Output Parameters

| | |
|---|---|
| `version` | The supported BGP version |

### Return Value

BGP_API_GET_SUCCESS when the function finds the version

## bgp_set_peer_admin_status

This function modifies the administration status of a peer. It either starts it (puts it in a running state) or stops it.

### Syntax

```
int
bgp_set_peer_admin_status (u_int32_t vr_id, int proc_id,
struct pal_in4_addr *addr, long flag);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | An integer that represents the BGP Process ID |
| `addr` | The address of the peer |
| `flag` | The current status of the peer |

### Return Value

BGP_API_SET_SUCCESS when the function modifies the peer's status

BGP_API_SET_ERROR when the function fails to set admin status (-1)

## bgp_get_peer_local_addr

This function returns the local address of the peer's BGP connection.

### Syntax

```
int
bgp_get_peer_local_addr (u_int32_t vr_id, int proc_id, struct pal_in4_addr *addr,
struct pal_in4_addr *out);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the peer |

### Output Parameters

| | |
|---|---|
| `out` | The local address of the peer's BGP connection. |

### Return Value

BGP_API_GET_SUCCESS when the call finds the peer's local address (0)

BGP_API_GET_ERROR when the function fails to get peer address (-1)

## bgp_set_next_peer_connect_retry_interval

This function modifies the time interval in seconds for the connect retry timer of the next peer. The suggested value for this timer is 120 seconds.

### Syntax

```
int
bgp_set_next_peer_connect_retry_interval (u_int32_t vr_id, int proc_id,
struct pal_in4_addr *addr, int tm);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter. |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the next peer |
| `tm` | Supplies the new time interval |

### Return Value

BGP_API_SET_SUCCESS when the call modifies the connect retry interval of the next peer (0)

BGP_API_SET_ERROR when the function fails to set next peer connect retry interval (-1)

## bgp_get_peer_hold_time

This function returns the time interval in seconds that the Hold timer has been established with the BGP peer. The value must be at least 3 seconds or zero (0), which means the Hold timer has not been established with the peer.

### Syntax

```
int
bgp_get_peer_hold_time (u_int32_t vr_id, int proc_id, struct pal_in4_addr *addr,
int *tm);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the peer |

### Return Value

| | |
|---|---|
| `tm` | The hold time interval in seconds |

### Return Value

BGP_API_GET_SUCCESS when the call finds the peer's hold time (0)

BGP_API_GET_ERROR when the function fails to get peer hold time (-1)

## bgp_set_peer_hold_time_configured

This function modifies the time interval in seconds for the hold time configured for this BGP speaker with the peer. The value must be at least 3 seconds or 0 (zero), which means the Hold timer has not been established with the peer. The suggested value for this timer is 90 seconds.

### Syntax

```
int
bgp_set_peer_hold_time_configured (u_int32_t vr_id, int proc_id,
struct pal_in4_addr *addr, int *tm);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the peer |
| `tm` | supplies the new time interval |

### Return Value

BGP_API_SET_SUCCESS when the call modifies the peer's hold time configured interval (0)

BGP_API_SET_ERROR when the call fails to modify the hold time interval (-1)

## bgp_set_peer_keep_alive_configured

This function modifies the time interval in seconds for the KeepAlive timer configured for this BGP speaker with the peer. If the value of this object is zero, no periodical KEEPALIVE messages are sent to the peer after the BGP connection has been established. The suggested value for this timer is 30 seconds.

### Syntax

```
int
bgp_set_peer_keep_alive_configured (u_int32_t vr_id, int proc_id,
struct pal_in4_addr *addr, int *tm);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the peer |
| `tm` | Supplies the new time interval |

### Return Value

BGP_API_SET_SUCCESS when the call modifies the peer's KeepAlive configured interval (0)

BGP_API_SET_ERROR when the function fails to configure peer hold time (-1)

## bgp_set_peer_min_route_advertisement_interval

This function modifies the time interval in seconds for the MinRouteAdvertisementInterval timer. The suggested value for this timer is 30 seconds.

### Syntax

```
int
bgp_set_peer_min_route_advertisement_interval (u_int32_t vr_id, int proc_id,
struct pal_in4_addr *addr, int *tm);
```

### Input Parameters

| | |
|---|---|
| `vr_id` | The Virtual Router ID. Its default value is 0. For a non-VR implementation, pass 0 for this parameter. |
| `proc_id` | The BGP Process ID |
| `addr` | The address of the peer |
| `tm` | Supplies the new time interval |

### Return Value

BGP_API_SET_SUCCESS when the call modifies the peer's MinRouteAdvertisementInterval timer (0)

BGP_API_SET_ERROR when the function fails to set peer min route advertisement interval (-1)

BGP_API_SET_ERR_INVALID_VALUE when tm value is invalid (-2)

BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER when the command is invalid for a peer-group member (-18)

BGP_API_SET_ERR_ALREADY_SET when the route advertisement value is already set (-77)

# Appendix A Return Values and Flags

This appendix lists the return values and flags that are used with the functions documented in this guide.



## Return Values

.

| Return Value | Cause of the Result |
|---|---|
| True | Success |
| False | Failure |
| BGP_API_SET_SUCCESS | The called function worked properly as specified |
| BGP_API_SET_ERROR | The called function did not work properly |
| BGP_API_SET_ERR_INVALID_VALUE | The scan time interval exceeds the allowed maximum |
| BGP_API_SET_ERR_INVALID_FLAG | The specified flag is invalid |
| GP_API_SET_ERR_INVALID_AS | The AS value is invalid |
| BGP_API_SET_ERR_INVALID_BGP | The BGP instance is equal to zero |
| BGP_API_SET_ERR_NO_CAP_CMD | The peer is not enabled with the graceful shutdown capability |
| BGP_API_SET_ERR_PEER_GROUP_MEMBER | The peer is a peer-group member and the peer-group already has AS configuration |
| BGP_API_SET_ERR_PEER_GROUP__UNCONFIGURED | The peer-group is not activated for the specified identifier and subsequent identifier |
| BGP_API_SET_ERR_PEER_GROUP_NO_REMOTE_AS | The peer-group does not have AS configuration |
| BGP_API_SET_ERR_PEER_GROUP_CANT_CHANGE | The peer already belongs to different peer-group |
| BGP_API_SET_ERR_PEER_GROUP_MISMATCH | The peer already belongs to different peer-group for a different Address Family configuration |
| BGP_API_SET_ERR_PEER_GROUP_PEER_TYPE_DIFFERENT | A try occurs to create an iBGP peer inside an eBGP peer-group or vice versa |
| BGP_API_SET_ERR_MULTIPLE_INSTANCE_NOT_SET | BGP multiple instance is not enabled |
| BGP_API_SET_ERR_AS_MISMATCH | BGP is already running under a different AS number |

| Return Value | Cause of the Result |
|---|---|
| BGP_API_SET_ERR_PEER_INACTIVE | The peer is not activated on specified I and SI |
| BGP_API_SET_ERR_INVALID_FOR_PEER_GROUP_MEMBER | The specified flag cannot be used for peer- group member peer |
| BGP_API_SET_ERR_PEER_GROUP_HAS_THE_FLAG | The peer is a peer-group member; need to change the peer-group configuration |
| BGP_API_SET_ERR_PEER_FLAG_CONFLICT | Conflicting flags are specified simultaneously: PEER_FLAG_STRICT_CAP_MATCH and PEER_FLAG_OVERRIDE_CAPABILITY |
| BGP_API_SET_ERR_PEER_GROUP_SHUTDOWN | Specification for no neighbor shutdown for the peer which belongs to peer-group, and the peer-group has a neighbor shutdown flag |
| BGP_API_SET_ERR_PEER_FILTER_CONFLICT | The distribute-list is already configured to the peer |
| BGP_API_SET_ERR_NOT_INTERNAL_PEER | The specified flag cannot be used for iBGP peer |
| BGP_API_SET_ERR_REMOVE_PRIVATE_AS | Trying to set the PEER_FLAG_REMOVE_PRIVATE_AS for iBGP peer |
| BGP_API_SET_ERR__UNCONFIGURED | The address family is not unconfigured |
| BGP_API_SET_ERR_SOFT_RECONFIG_UNCONFIGURED | The peer does not send route-refresh capability and soft-reconfiguration inbound is not configured for the peer |
| BGP_API_SET_ERR_SOFT_RECONFIG_UNCONFIGURED | The peer does not send route-refresh capability and soft-reconfiguration inbound is not configured for the peer |
| BGP_API_SET_ERR__UNCONFIGURED | The address family is not configured |
| BGP_API_SET_ERR_INSTANCE_MISMATCH | The AS number is different in the current configuration |
| BGP_API_SET_ERR_MALFORMED_ARG | The community list type is EXTCOMMUNITY_LIST_STANDARD |
| BGP_API_SET_ERR_CLIST_DEFINE_CONFLICT | There is a Community name conflict with the previously defined community list |
| BGP_API_SET_ERR_PEER_GROUP__INVALID | Invalid combination of address families for the peer-group |
| BGP_API_SET_ERR_MULT_INST_DEL_CONFIG | Trying to delete multi instance configurations |
| BGP_API_SET_ERR_PEER_CONFIG_IN_ANOTHER_INST | The peer is configured in another instance |
| BGP_API_SET_ERR_UNSUP_VPNVF_CONF | Unable to activate VPNV4 family for EBGP peer |
| BGP_API_SET_ERR_INVALID | The SI is not Unicast |
| BGP_API_IP_NOT_IN_SAME_SUBNET | The interface IP address and neighbor IP are not in same subnet |
| BGP_API_INVALID_INTERFACE_NAME | The interface name is invalid |
| BGP_API_SET_ERR_ALREADY_EXTASNCAP. | The extended ASN capability is already enabled |

| Return Value | Cause of the Result |
|---|---|
| BGP_API_SET_ERR_NO_EXTASNCAP when (-54) | An extended ASN capability is not enabled, and an attempt is made to configure BGP with 4-octet non-mappable |
| BGP_API_SET_ERR_NONMAPPABLE | The BGP is configured with 4-octet non-mappable ASN and an attempt is made to disable extended ASN capability |
| BGP_API_SET_ERR_INFINITE_HOLD_TIME_VALUE | The specified hold time is infinite value |
| BGP_API_SET_WARN_HOLD_AND_KEEPALIVE_INVALID | The configured holdtime is not equal to 0 and the hold time less than the 3 times of keepalive time |
| BGP_API_SET_ERR_INVALID_HOLD_TIME | Hold time value is not equal to 0 and less than 3 |
| BGP_API_INVALID_ROUTE_NODE | The route node is invalid |
| BGP_API_SET_ERR_AUTO_SUMMARY_ENABLED | Auto summary is enabled |
| BGP_API_SET_ERR_AUTO_SUMMARY_DISABLED | Auto summary is disabled |
| BGP_API_SET_ERR_INVALID_REMOTEASN | The remote AS number is Invalid |
| BGP_API_NHT_NOT_ENABLED_SET_ERR | Nexthop tracking is not enabled |
| BGP_API_SET_ERR_NHT_DISABLE_RIB_SCAN | Disabling RIB-SCAN is not allowed: NHT is enabled |
| BGP_API_SET_ERR_ALREADY_SET | The configurations are already set |
| BGP_API_SET_ERR_ADJ_OUT_DYNAMIC | Trying to change this option while the BGP instance is running |
| BGP_API_SET_ERR_HOLD_LESS_EQUAL_KEEPALIVE | The hold time is not equal to 0 and less than or equal to keep alive time |

## Adress Family Flags

The following table lists the address family (AF) configuration flags that are used in external BGP functions.

| Flag | Description |
|---|---|
| PEER_FLAG_SEND_COMMUNITY | send-community |
| PEER_FLAG_SEND_EXT_COMMUNITY | send-community ext. |
| PEER_FLAG_NEXTHOP_SELF | next-hop-self |
| PEER_FLAG_REFLECTOR_CLIENT | reflector-client |
| PEER_FLAG_RSERVER_CLIENT | route-server-client |
| PEER_FLAG_SOFT_RECONFIG | soft-reconfiguration |
| PEER_FLAG_AS_PATH_UNCHANGED | transparent-as |

| Flag | Description |
|---|---|
| PEER_FLAG_NEXTHOP_UNCHANGED | transparent-next-hop |
| PEER_FLAG_MED_UNCHANGED | transparent-med |
| PEER_FLAG_DEFAULT_ORIGINATE | default-originate |
| PEER_FLAG_REMOVE_PRIVATE_AS | remove-private-as |
| PEER_FLAG_ALLOWAS_IN | set allowas-in |
| PEER_FLAG_ORF_PREFIX_SM | orf capability send-mode |
| PEER_FLAG_ORF_PREFIX_RM | orf capability receive-mode |
| PEER_FLAG_MAX_PREFIX_WARNING | maximum prefix warning-only |
| PEER_FLAG_AS_OVERRIDE | AS override |
| PEER_FLAG_SITE_ORIGIN | set site-origin |
| PEER_FLAG_GRST_CAPABILITY | graceful-restart |
| PEER_FLAG_EBGP_VPN_ALLOW | Allow EBGP VPN |

# Appendix B Source Code

This appendix lists the source files of the Border Gateway Protocol module. Only external functions are covered in this document; other functions are outside the scope of this manual.

The BGP source code contains the following files:

| | | | |
|---|---|---|---|
| BGP4-MIB.txt | bgp_damp.h | bgp_fsm.c | bgp_ptree.h |
| bgp_advertise.c | bgpd.c | bgp_fsm.h | bgp_regex.c |
| bgp_advertise.h | bgpd.conf.sample | bgp_incl.h | bgp_regex.h |
| bgp_api.c | bgpd.conf.sample2 | bgp_main.c | bgp_route.c |
| bgp_api.h | bgp_debug.c | bgp_md5.c | bgp_route.h |
| bgp_as4path.c | bgp_debug.h | bgp_md5.h | bgp_routemap.c |
| bgp_as4path.h | bgp_decode.c | bgp_mpls.c | bgp_show.c |
| bgp_aspath.c | bgp_decode.h | bgp_mpls.h | bgp_snmp.c |
| bgp_aspath.h | bgpd.h | bgp_mplsvpn.c | bgp_snmp.h |
| bgp_attr.c | bgp_dump.c | bgp_mplsvpn.h | bgp_vrf.c |
| bgp_attr.h | bgp_dump.h | bgp_network.c | bgp_vrf.h |
| bgp_cli.c | bgp_ecommunity.c | bgp_network.h | Makefile |
| bgp_clist.c | bgp_ecommunity.h | bgp_nexthop.c | Rules.dir |
| bgp_clist.h | bgp_encode.c | bgp_nexthop.h | |
| bgp_community.c | bgp_encode.h | bgp_nsm.c | |
| bgp_community.h | bgp_filter.c | bgp_nsm.h | |
| bgp_damp.c | bgp_filter.h | bgp_ptree.c | |

# Index

## G



## I



## L



## M



## N



## O



## P



## R

## S



## U



## V



## W