Chapter 33

BGP Configuration Guidelines

To configure the Border Gateway Protocol (BGP), you can include the following statements. Three portions of the bgp statement—those in which you configure global BGP, group-specific, and peer-specific options—contain many of the same statements. The following simplified version of the bgp statement omits these repeated statements to present a high-level, readable overview:

```
protocols {
    bgp {
        numerous global BGP statements
        group group-name {
            peer-as autonomous-system;
            type type;
            [network/mask-length ];
            numerous group-specific statements;
            neighbor address {
                numerous peer-specific statements;
            }
        }
    }
}
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

For a list of global BGP statements, see “Defining BGP Global Properties” on page 545. For a list of group-specific statements, see “Defining Group Properties” on page 550. For a list of peer-specific statements, see “Defining Peer Properties” on page 552.

NOTE: Changing configuration statements that affect BGP peerings, such as enabling or disabling remove-private or renaming a BGP group, resets the BGP sessions. Changes that affect BGP peerings should only be made when resetting a BGP session is acceptable.

Many of the global BGP, group-specific, and peer-specific statements are identical. For statements that you can configure at more than one level in the hierarchy, the more-specific statement overrides the less-specific statement. That is, a group-specific statement overrides a global BGP statement, and a peer-specific statement overrides a global BGP or group-specific statement.

By default, BGP is disabled.

This chapter describes the following tasks for configuring BGP:

Minimum BGP Configuration

For BGP to run on the router, you must define the local autonomous system (AS) number, configure at least one group, and include information about at least one peer in the group (the peer's IP address and AS number). There are several ways you can configure this information; a few are shown in this section.

Configure a BGP group, specify the group type, and configure an explicit peer:

```
[edit]
routing-options {
    autonomous-system autonomous-system;
}
protocols {
    bgp {
        group group-name {
            peer-as autonomous-system;
            type type;
            neighbor address;
        }
    }
}
```

Configure a BGP group and type and allow all BGP systems to be peers:

```
[edit]
routing-options {
    autonomous-system autonomous-system;
}
protocols {
    bgp {
        group group-name {
            type type;
            peer-as autonomous-system;
            all;
        }
    }
}
```

NOTE: When you configure BGP on an interface, you must also include the family inet statement at the [edit interfaces *interface-name* unit *logical-unit-number*] hierarchy level. For more information about the family inet statement, see the *JUNOS Network Interfaces Configuration Guide*.

Enabling BGP

To enable BGP on the router, perform the following tasks:

- Specifying the Local Router's AS Number on page 544 (required)
- Defining an AS Confederation and Its Members on page 545 (optional)
- Assigning a BGP Identifier on page 545 (optional)
- Defining BGP Global Properties on page 545 (optional)
- Defining BGP Groups and Peers on page 547 (required)

For examples of enabling BGP, see the following sections:

- Example: Defining a Large Number of Group with Static Peers on page 548
- Example: Defining a Small Number of Groups with Static Peers for Better Scalability on page 549

Specifying the Local Router's AS Number

Each router running BGP must be configured with its AS number. This number is included in the local AS number field in BGP open messages, which are sent between BGP peers to establish a connection.

To specify an AS number, include the autonomous-system statement:

```
autonomous-system autonomous-system <loops number>;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

You must specify an AS number to enable BGP.

For more information about configuring the AS number, see “Configuring the AS Number” on page 102.

Defining an AS Confederation and Its Members

To enable the local system to participate as a member of an AS confederation, you must define the AS confederation identifier and specify the AS numbers that are members of the confederation. To do this, include the confederation statement:

```
confederation confederation-autonomous-system members
    [ autonomous-systems ];
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Defining an AS confederation and its members is optional.

For more information about configuring confederations, see “Configuring AS Confederation Members” on page 103.

Assigning a BGP Identifier

Each router running BGP must have a BGP identifier. This identifier is included in the BGP identifier field of open messages, which are sent between two BGP peers when establishing a BGP session.

Explicitly assigning a BGP identifier is optional. If you do not assign one, the IP address of the first interface encountered in the router is used.

To assign a BGP identifier explicitly, include the router-id statement:

```
router-id address;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Assigning a BGP identifier is optional.

For more information, see “Configuring the Router Identifier” on page 102.

Defining BGP Global Properties

To define BGP global properties, which apply to all BGP groups and peers, include one or more of the following statements. These statements are explained in separate sections.

```
advertise-inactive;
advertise-peer-as;
authentication-algorithm algorithm;
authentication-key key;
authentication-key-chain key-chain;
```

```
cluster cluster-identifier;
damping;
description text-description;
disable;
family {
    (iso-vpn | inet | inet6 | inet-vpn | inet6-vpn | l2-vpn) {
        (any | multicast | unicast | signaling) {
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | minutes)>;
            }
            rib-group group-name;
        }
        labeled-unicast {
            aggregate-label {
                community community-name:
            }
            explicit-null {
                connected-only;
            }
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | minutes)>;
            }
            resolve-vpn;
            rib inet.3;
            rib-group group-name;
        }
    }
    route-target {
        advertise-default;
        external-paths number;
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
    }
    signaling {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
    }
}
export [ policy-names ];
graceful-restart {
    disable;
    restart-time seconds;
    stale-routes-time seconds;
}
hold-time seconds;
import [ policy-names ];
include-mp-next-hop;
ipsec-sa ipsec-sa;
keep (all | none);
local-address address;
local-as autonomous-system <private>;
```

```
local-preference local-preference;
log-updown;
metric-out (metric | minimum-igp <offset> | igp <offset>);
multihop <ttl-value>;
no-advertise-peer-as;
no-aggregator-id;
no-client-reflect;
out-delay seconds;
passive;
path-selection {
    (always-compare-med | cisco-non-deterministic | external-router-id);
    med-plus-igp {
        igp-multiplier number;
        med-multiplier number;
    }
}
peer-as autonomous-system;
preference preference;
remove-private;
tcp-mss segment-size;
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
        <(world-readable | no-world-readable)>;
    flag flag <flag-modifier> <disable>;
}
vpn-apply-export;
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

You must configure BGP global properties to enable BGP.

Defining BGP Groups and Peers

A BGP system must know which routers are its peers (neighbors). You define the peer relationships explicitly by configuring the neighboring routers that are the peers of the local BGP system. After peer relationships have been established, the BGP peers exchange update messages to advertise network reachability information.

You arrange BGP routers into groups of peers. Different peer groups must have different group types, AS numbers, or router reflector cluster identifiers.

Each group must contain at least one peer.

To define BGP groups and peers, see the following sections:

- Defining a Group with Static Peers on page 548
- Defining a Group with Dynamic Peers on page 549
- Defining the Group Type on page 550
- Specifying the Peer's AS Number on page 550
- Defining Group Properties on page 550

Defining a Group with Static Peers

To define a BGP group that recognizes only the specified BGP systems as peers, statically configure all the system's peers by including one or more neighbor statements. The peers on at least one side of each BGP connection must be configured statically. The peer neighbor's address can be either an IPv6 or IPv4 address.

```
group group-name {
    peer-as autonomous-system;
    type type;
    neighbor address;              # One "neighbor" statement for each peer
}
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

As the number of external BGP (EBGP) groups increases, the ability to support a large number of BGP sessions may become a scaling issue. The preferred way to configure a large number of BGP neighbors is to configure a few groups consisting of multiple neighbors per group. Supporting fewer EBGP groups generally scales better than supporting a large number of EBGP groups. This becomes more evident in the case of hundreds of EBGP groups when compared with a few EBGP groups with multiple peers in each group. The following examples illustrate this point.

Example: Defining a Large Number of Group with Static Peers

Enable BGP and define three EBGP groups that recognize BGP systems in AS 56, AS 57, and AS 58 as peers:

```
[edit]
routing-options {
    autonomous-system 23;
}
protocols {
    bgp {
      group G1 {
        type external;
        peer-as 56;
        neighbor 10.0.0.1;
      }
      group G2 {
        type external;
        peer-as 57;
        neighbor 10.0.10.1;
      }
```

```
        group G3 {
            type external;
            peer-as 58;
            neighbor 10.0.20.1;
        }
    }
}
```

Example: Defining a Small Number of Groups with Static Peers for Better Scalability

For improved scalability, configure only one EBGP group consisting of the three BGP neighbors:

```
[edit]
routing-options {
    autonomous-system 23;
}
protocols {
    bgp {
        group G {
            type external;
                neighbor 10.0.0.1 {
                    peer-as 56;
                }
                neighbor 10.0.10.1 {
                    peer-as 57;
                }
                neighbor 10.0.20.1 {
                    peer-as 58;
                }
        }
    }
}
```

Defining a Group with Dynamic Peers

To define a BGP group in which the local system's peers are dynamic and change over time, include the allow statement. To recognize all BGP systems as peers, include the allow-all statement. To recognize BGP systems within specified address ranges, specify a set of addresses in the allow *network/mask-length* statement. These addresses can be IPv6 or IPv4 addresses.

```
group group-name {
    peer-as autonomous-system;
    type type;
    (allow [ network/mask-length... ] | all);
}
```

NOTE: You cannot define a BGP group with dynamic peers with authentication enabled.

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

Defining the Group Type

When configuring a BGP group, you can indicate whether the group is an internal BGP (IBGP) group or an external BGP (EBGP) group. All peers in an IBGP group are in the same AS, while peers in an EBGP group are in different ASs and normally share a subnet.

To configure an IBGP group, which allows intra-AS BGP routing, include the following form of the type statement:

```
type internal;
```

To configure an EBGP group, which allows inter-AS BGP routing, include the following form of the type statement:

```
type external;
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

Specifying the Peer's AS Number

When configuring a peer, you must specify the peer system's AS. To do this, include the peer-as statement:

```
peer-as autonomous-system;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

For EBGP, the peer is in another AS, so the AS number you specify in the peer-as statement must be different from the local router's AS number, which you specify in the autonomous-system statement. For IBGP, the peer is in the same AS, so the two AS numbers that you specify in the autonomous-system and peer-as statements must be the same.

Defining Group Properties

To define group-specific properties, include one or more of the following statements. For more information, see "Summary of BGP Configuration Statements" on page 605.

```
advertise-inactive;
advertise-peer-as;
[ network/mask-length ];
as-override;
authentication-algorithm algorithm;
authentication-key key;
authentication-key-chain key-chain;
cluster cluster-identifier;
damping;
description text-description;
export [ policy-names ];
```

```
family {
    (inet | inet6 | inet-vpn | inet6-vpn | l2-vpn) {
        (any | multicast | unicast | signaling) {
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
            }
            rib-group group-name;
        }
        flow {
            no-validate policy-name;
        }
        labeled-unicast {
            explicit-null {
                connected-only;
            }
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
            }
            resolve-vpn;
            rib inet.3;
            rib-group group-name;
        }
    }
    route-target {
        advertise-default;
        external-paths number;
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
        }
    }
    signaling {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
    }
}
graceful-restart {
    disable;
    restart-time seconds;
    stale-routes-time seconds;
}
hold-time seconds;
import [ policy-names ];
ipsec-sa ipsec-sa;
keep (all | none);
local-address address;
local-as autonomous-system <private>;
local-preference local-preference;
log-updown;
metric-out (metric | minimum-igp <offset> | igp <offset>);
mtu-discovery;
multihop <ttl-value>;
```

```
multipath {
    multiple-as;
}
neighbor address {
    peer-specific options;
}
no-advertise-peer-as;
no-aggregator-id;
no-client-reflect;
out-delay seconds;
passive;
peer-as autonomous-system;
preference preference;
protocol protocol;
remove-private;
tcp-mss segment-size;
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
      <(world-readable | no-world-readable)>;
    flag flag <flag-modifier> <disable>;
}
type type;
vpn-apply-export;
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

Defining Peer Properties

When you use the neighbor statement to configure BGP peers statically, you also can define peer-specific properties. For more information, see "Summary of BGP Configuration Statements" on page 605.

```
advertise-inactive;
advertise-peer-as;
as-override;
authentication-algorithm algorithm;
authentication-key key;
authentication-key-chain key-chain;
cluster cluster-identifier;
damping;
description text-description;
export [ policy-names ];
family {
    (iso-vpn | inet | inet6 | inet-vpn | inet6-vpn | l2-vpn) {
      (any | multicast | unicast | signaling) {
        prefix-limit {
          maximum number;
          teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
        }
        rib-group group-name;
      }
      flow {
        no-validate policy-name;
      }
```

```
        labeled-unicast {
            explicit-null {
                connected-only;
            }
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
            }
            resolve-vpn;
            rib inet.3;
            rib-group group-name;
        }
    }
    route-target {
        advertise-default;
        external-paths number;
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
        }
    }
    signaling {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
    }
}
graceful-restart {
    disable;
    restart-time seconds;
    stale-routes-time seconds;
}
hold-time seconds;
import [ policy-names ];
ipsec-sa ipsec-sa;
keep (all | none);
local-address address;
local-as autonomous-system <private>;
local-interface interface-name;
local-preference preference;
log-updown;
metric-out (metric | minimum-igp <offset> | igp <offset>);
mtu-discovery
multihop <ttl-value>;
multipath {
    multiple-as;
}
no-advertise-peer-as;
no-aggregator-id;
no-client-reflect;
out-delay seconds;
passive;
peer-as autonomous-system;
preference preference;
tcp-mss segment-size;
```

```
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
      <(world-readable | no-world-readable)>;
    flag flag <flag-modifier> <disable>;
}
vpn-apply-export;
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

Examples: Enabling BGP

Enable BGP and define an EBGP group that recognizes all BGP systems in AS 56 as peers:

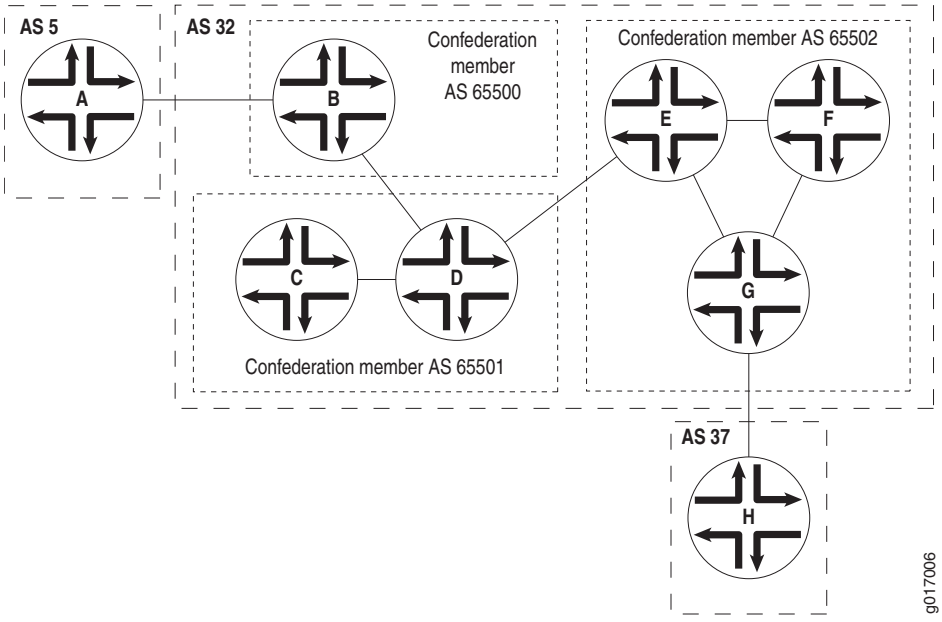
```
[edit]
routing-options {
    autonomous-system 23;
}
protocols {
    bgp {
      group 23 {
        type external;
        peer-as 56;
        0.0.0.0/0;
      }
    }
}
```

Enable BGP and define an IBGP group that recognizes only the specified addresses as BGP peers.

```
[edit]
routing-options {
    autonomous-system 23;
    router-id 10.0.0.1;
}
protocols {
    bgp {
      group 23 {
        type internal;
        peer-as 23;
        neighbor 10.0.0.2;
        neighbor 10.0.0.3;
      }
    }
}
```

Configure a BGP confederation. Figure 9 illustrates the confederation topology used in this example. For AS 32 to be a valid confederation, all routers in the AS must be members of the confederation. For example, Router B must have a confederation member AS number as well as a confederation AS number. Within a confederation, the links between the confederation member ASs must be EBGP links, not IBGP links.

Figure 9: Example: BGP Confederation Topology



On Router A:

```
[edit]
routing-options {
    autonomous-system 5;
}
protocols {
    bgp {
      group AtoB {
        type external;
        peer-as 32;
        neighbor 10.0.0.2;
      }
    }
}
```

On Router B:

```
[edit]
routing-options {
    autonomous-system 65500;
    confederation 32 members [65500 65501 65502];
}
```

```
protocols {
    bgp {
      group BtoA {
        type external;
        peer-as 5;
        neighbor 10.0.0.1;
      }
      group BtoD {
        type external;
        peer-as 65501;
        neighbor 10.0.10.2;
      }
    }
}
```

On Router C:

```
[edit]
routing-options {
    autonomous-system 65501;
    confederation 32 members [65500 65501 65502];
}
protocols {
    bgp {
      group CtoD {
        type internal;
        neighbor 10.0.10.3;
      }
    }
}
```

On Router D:

```
[edit]
routing-options {
    autonomous-system 65501;
    confederation 32 members [65500 65501 65502];
}
protocols {
    bgp {
      group DtoC {
        type internal;
        neighbor 10.0.10.1;
      }
      group DtoB {
        type external;
        peer-as 65500;
        neighbor 10.0.10.1;
      }
      group DtoE {
        type external;
        peer-as 65502;
        neighbor 10.0.30.1;
      }
    }
}
```

On Router E:

```
[edit]
routing-options {
    autonomous-system 65502;
    confederation 32 members [65500 65501 65502];
}
protocols {
    bgp {
        group EtoD {
            type external;
            peer-as 65501;
            neighbor 10.0.10.4;
        }
        group EtoFandG {
            type internal;
            neighbor 10.0.30.2;
            neighbor 10.0.30.5;
        }
    }
}
```

On Router F:

```
[edit]
routing-options {
    autonomous-system 65502;
    confederation 32 members [65500 65501 65502];
}
protocols {
    bgp {
        group FtoEandG {
            type internal;
            neighbor 10.0.30.3;
            neighbor 10.0.30.7;
        }
    }
}
```

On Router G:

```
[edit]
routing-options {
    autonomous-system 65502;
    confederation 32 members [65500 65501 65502];
}
protocols {
    bgp {
        group GtoH {
            type external;
            peer-as 37;
            neighbor 10.0.40.1;
        }
        group GtoEandF {
            type internal;
            neighbor 10.0.30.4;
            neighbor 10.0.30.5;
        }
    }
}
```

On Router H:

```
[edit]
routing-options {
    autonomous-system 37;
}
protocols {
    bgp {
        group HtoG {
            type external;
            peer-as 32;
            neighbor 10.0.30.8;
        }
    }
}
```

Modifying the Hold-Time Value

The hold time is the maximum number of seconds allowed to elapse between successive keepalive or update messages that BGP receives from a peer. When establishing a BGP connection with the local router, a peer sends an open message, which contains a hold-time value. BGP on the local router uses the smaller of either the local hold-time value or the peer's hold-time value received in the open message as the hold time for the BGP connection between the two peers.

To modify the hold-time value on the local BGP system, include the hold-time statement:

```
hold-time seconds;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

The default hold-time value is 90 seconds.

The hold time is three times the interval at which keepalive messages are sent.

Configuring MTU Discovery

You can configure Transmission Control Protocol (TCP) path maximum transmission unit (MTU) discovery. MTU discovery improves convergence times for internal BGP sessions.

To configure MTU discovery, include the mtu-discovery statement:

```
mtu-discovery;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring Graceful Restart

Graceful restart is disabled by default. You can globally enable graceful restart for all routing protocols at the [edit routing-options] or [edit logical-routers *logical-router-name* routing-options] hierarchy levels.

To configure graceful restart specifically for BGP, include the graceful-restart statement:

```
graceful-restart {
    restart-time;
    stale-routes-time;
}
```

For a list of hierarchy levels at which you can configure this statements see the statement summary section for this statement.

NOTE: Configuring graceful restart for BGP resets the BGP peer routing statistics to zero.

To disable graceful restart for BGP, specify the disable statement. To configure a time period to complete restart, specify the restart-time statement. To configure a time period over which to keep stale routes during a restart, specify the stale-routes-time statement.

Advertising an Explicit Null Label

You can advertise an explicit null label (label 0) out of the egress for a label-switch path (LDP). By default, the router advertises label 3. Enabling explicit null allows the router to send out label 0. Advertising explicit null labels is used for peers in the same BGP group.

Configure the labeled-unicast statement with the explicit-null option. As with regular BGP configuration, the family statement can be specified.

Configure the labeled-unicast statement as follows:

```
family inet {
    labeled-unicast {
        aggregate-label {
            community community-name:
        }
        explicit-null {
            connected-only;
        }
    }
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Specify the connected-only statement to advertise explicit null labels between connectly routes only (direct routes and loopback routes).

NOTE: The connected-only statement is required to advertise explicit null labels.

NOTE: Explicit null labels are supported for the IPv4 (inet) family only.

Configuring Aggregate Labels for VPNs

Aggregate labels for VPNs allow a Juniper Networks routing platform to aggregate a set of incoming labels (labels received from a peer router) into a single forwarding label that is selected from the set of incoming labels. The single forwarding label corresponds to a single next hop for that set of labels.

For a set of labels to share a single forwarding label, they must belong to the same forwarding equivalence class (FEC). The labeled packets must have the same destination egress interface.

To configure aggregate labels for VPNs, include the aggregate-label statement:

```
aggregate-label {
    community community-name;
}
```

For a list of hierarchy levels at which you can include the aggregate-label statement, see the statement summary for this statement.

Configuring Authentication

All BGP protocol exchanges can be authenticated to guarantee that only trusted routers participate in the AS's routing. By default, authentication is disabled on the router. You can configure MD5 authentication on the router. The MD5 algorithm creates an encoded checksum that is included in the transmitted packet. The receiving router uses an authentication key (password) to verify the packet's MD5 checksum.

To configure an MD5 authentication key, include the authentication-key statement:

```
authentication-key key;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

If you configure authentication for all peers, each individual peer in that group inherits the group's authentication.

The key (password) can be up to 126 characters long. Characters can include any ASCII strings. If you include spaces, enclose all characters in quotation marks (double quotes).

You can update MD5 authentication keys without resetting any BGP peering sessions. This is referred to as hitless authentication key rollover. Hitless authentication key rollover uses authentication key chains, which consist of the authentication keys that are being updated.

Hitless authentication key rollover also allows users to choose the algorithm through which authentication is established. The user associates a key chain and an authentication algorithm with a BGP neighboring session. The key chain includes multiple keys. Each key contains an identifier and a secret. The key is also configured with a unique start time and an end time.

The sending peer chooses the active key based on the system time. The receiving peer determines the key with which it authenticates based upon the incoming key identifier.

To configure the authentication key, include the key-chain statement at the [edit security authentication-key-chains] hierarchy level, and specify the key option to create a key chain consisting of several authentication keys.

```
[edit security]
authentication-key-chains {
    key-chain key-chain-name {
        key key {
            secret secret-data;
            start-time yyyy-mm-dd.hh:mm:ss;
        }
    }
}
```

You can configure multiple keys within the key chain.

Each key within a key chain must be identified by a unique integer value configured in the key statement. The range of valid identifier values is from 0 through 63. Each key must specify a secret. This secret can be entered in either encrypted or plain text format in the secret statement. It is always displayed in encrypted format.

Each key must specify a start time with the start-time statement. Start times are specified in the local time zone for a router and must be unique within the key chain.

For more information on configuring authentication key chains, see the *JUNOS System Basics Configuration Guide*.

To apply an authentication key chain to the router, include the authentication-key-chain statement:

```
authentication-key-chain key-chain;
```

To specify the authentication algorithm type to use for key chains, include the authentication-algorithm statement:

```
authentication-algorithm algorithm;
```

You can choose either md5 or hmac-sha-1-96 as the type of algorithm.

NOTE: BGP authentication is not supported with promiscuous mode BGP sessions. If you include the allow statement, you cannot include authentication-key or authentication-key-chain at the same hierarchy level or any higher hierarchy level. When configuring authentication for all peers in a group, you cannot include the allow statement in the configuration because BGP keys require a destination address.

For a list of hierarchy levels at which you can include the previous statements, see the statement summary for those statements.

Applying IPSec Security Association

You can apply IPSec to BGP traffic. IPSec is a protocol suite used for protecting IP traffic at the packet level. IPSec is based on security associations (SAs). A security association is a simplex connection that provides security services to the packets carried by the SA. After configuring the security association, you can apply the SA to BGP peers.

To apply a security association, include the ipsec-sa statement:

```
ipsec-sa ipsec-sa;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement. The security association is identified by the SA name.

NOTE: Tunnel mode requires the ES PIC.

In transport mode, the JUNOS software does not support authentication header (AH) or encapsulating security payload (ESP) header bundles.

The JUNOS software supports only BGP in transport mode.

A more specific security association overrides a less general SA. For example, if a specific SA is applied to a specific peer, that SA overrides the SA applied to the whole peer group.

For more detailed information about configuring IPSec security associations, see the *JUNOS System Basics Configuration Guide*.

Opening a Peer Connection Passively

You can configure a router not to send Open requests to a peer. Once you configure the router to be passive, the router does not originate the TCP connection. However, when the router receives a connection from the peer and an Open message, it replies with another BGP Open message. Each router declares its own capabilities.

To configure the router so that it does not send Open requests to a peer, include the passive statement:

```
passive;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring the Local IP Address

You can specify the address of the local end of a BGP session. You generally do this to explicitly configure the system's IP address from BGP's point of view. This IP address can be either an IPv6 or IPv4 address. Typically, an IP address is assigned to a loopback interface, and that IP address is configured here. This address is used to accept incoming connections to the peer and to establish connections to the remote peer. To assign a local address, include the local-address statement:

```
local-address address;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

If you include the default-address-selection statement in the configuration, the software chooses the system default address as the source for most locally generated IP packets. For more information, see the *JUNOS System Basics Configuration Guide*. For protocols in which the local address is unconstrained by the protocol specification, for example IBGP and multihop EBGP, if you do not configure a specific local address when configuring the protocol, the local address is chosen using the same methods as other locally generated IP packets.

Configuring the Multiple Exit Discriminator Metric

The BGP multiple exit discriminator (MED, or MULTI_EXIT_DISC) is an optional path attribute that can be included in BGP update messages. This attribute is used on external BGP links (that is, on inter-AS links) to select among multiple exit points to a neighboring AS. The MED attribute has a value that is referred to as a *metric*. If all other factors in determining an exit point are equal, the exit point with the lowest metric is preferred.

If a MED is received over an external BGP link, it is propagated over internal links to other BGP systems within the AS.

BGP update messages include a MED metric if the route was learned from BGP and already had a MED metric associated with it, or if you configure the MED metric in the configuration file in one of the following ways:

- Defining a MED Metric Directly on page 564
- Using Routing Policy to Define a MED Metric on page 565

For configuration examples, see "Examples: Configuring the MED Metric" on page 565.

A MED metric is advertised with a route according to the following general rules:

- A more specific metric overrides a less specific metric. That is, a group-specific metric overrides a global BGP metric and a peer-specific metric overrides a global BGP or group-specific metric.
- A metric defined with routing policy overrides a metric defined with the metric-out statement.
- If any metric is defined, it overrides a metric received in a route.
- If the received route did not have an associated MED metric, and if you did not explicitly configure a metric, no metric is advertised.

For a description of the algorithm used to determine the active path, see "How the Active Route Is Determined" on page 7.

Defining a MED Metric Directly

To directly configure a MED metric to advertise in BGP update messages, include the metric-out statement:

metric-out (*metric* | minimum-igp <*offset*> | igp <*offset*>);

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

metric is the primary metric on all routes sent to peers. It can be a value in the range from 0 through $2^{32} - 1$.

Specify minimum-igp to set the metric to the minimum metric value calculated in the IGP to get to the BGP next hop. If a newly calculated metric is greater than the minimum metric value, the metric value remains unchanged. If a newly calculated metric is lower, the metric value is lowered to that value.

Specify igp to set the metric to the most recent metric value calculated in the IGP to get to the BGP next hop.

Specify a value for <*offset*> to increase or decrease the metric that is used from the metric value calculated in the IGP. The metric value is offset by the value specified. The metric calculated in the IGP (by specifying either igp or igp-minimum) is increased if the <*offset*> value is positive. The metric calculated in the IGP (by specifying either igp or igp-minimum) is decreased if the <*offset*> value is negative.

offset can be a value in the range from -2^{31} through $2^{31} - 1$. Note that the adjusted metric can never go below 0 or above $2^{32} - 1$.

Using Routing Policy to Define a MED Metric

To use routing policy to define a MED metric to advertise, define the routing policy with the policy-statement statement at the [edit policy-options] hierarchy level, and then apply the filter using the import and export statements when configuring BGP.

When defining the routing policy filter, include an action that specifies the desired metric value:

```
policy-statement policy-name {
    term term-name {
        from {
            match-conditions;
            route-filter destination-prefix match-type <actions>;
            prefix-list name;
        }
        to {
            match-conditions;
        }
        then actions;
    }
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

For information about defining routing policy, see the *JUNOS Policy Framework Configuration Guide*. For information about applying filters in BGP, see "Configuring BGP Routing Policy" on page 594.

Examples: Configuring the MED Metric

Set the MED metric to 20 for all routes advertised in BGP update messages except for those sent to the peer system 192.168.0.1; the MED for this peer is 10:

```
[edit]
routing-options {
    router-id 10.0.0.1;
    autonomous-system 23;
}
protocols {
    bgp {
        metric-out 20;
        group 23 {
            type external;
            peer-as 56;
            neighbor 192.168.0.1 {
                traceoptions {
                    file bgp-log-peer;
                    flag packets;
                }
                log-updown;
                metric-out 10;
            }
        }
    }
}
```

Set the MED metric to 20 for all routes from a particular community:

```
[edit]
routing-options {
    router-id 10.0.0.1;
    autonomous-system 23;
}
policy-options {
    policy-statement from-otago {
        from community otago;
        then metric 20;
    }
    community otago members [56:2379 23:46944];
}
protocols {
    bgp {
        import from-otago;
        group 23 {
            type external;
            peer-as 56;
            neighbor 192.168.0.1 {
                traceoptions {
                    file bgp-log-peer;
                    flag packets;
                }
                log-updown;
            }
        }
    }
}
```

Controlling the Aggregator Path Attribute

The JUNOS implementation of BGP performs route aggregation, which is the process of combining the characteristics of different routes so that only a single route is advertised. Aggregation reduces the amount of information that BGP must store and exchange with other BGP systems.

BGP adds the aggregator path attribute to BGP update messages. This attribute contains the local system's AS number and IP address (router ID).

To prevent different routers within an AS from creating aggregate routes that contain different AS paths, set the IP address in the aggregator path attribute to 0 by including the no-aggregator-id statement:

```
no-aggregator-id;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Choosing the Protocol Used to Determine the Next Hop

By default, BGP uses the active routes determined from all IGPs when resolving routes to next hops. To limit the IGPs that BGP uses, include the protocol statement, specifying the protocol as isis or ospf:

```
protocol protocol;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring an EBGP Multihop Session

If an EBGP peer is more than one hop away from the local router, you must specify the next hop to the peer so that the two systems can establish a BGP session. This type of session is called a *multihop* BGP session. To configure a multihop session, include the multihop statement:

```
multihop {
    <ttl-value>;
    no-nexthop-change;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

To configure the maximum time-to-live (TTL) value for the TTL in the IP header of BGP packets, specify *ttl-value*. If you do not specify a TTL value, the system's default maximum TTL value is used. To specify not to change the BGP next-hop value for route advertisements, specify the no-nexthop-change option.

Configuring the BGP Local Preference

Internal BGP sessions use a metric called the *local preference*, which is carried in internal BGP update packets in the path attribute LOCAL_PREF. This metric indicates the degree of preference for an external route. The route with the highest local preference value is preferred.

The LOCAL_PREF path attribute is always advertised to internal BGP peers and to neighboring confederations. It is never advertised to external BGP peers. The default behavior is to not modify the LOCAL_PREF path attribute if it is present.

NOTE: The LOCAL_PREF path attribute applies at export time only.

By default, if a received route contains a LOCAL_PREF path attribute value, the value is not modified. If a BGP route is received without a LOCAL_PREF attribute, the route is handled locally (that is, it is stored in the routing table and advertised by BGP) as if it were received with a LOCAL_PREF value of 100. A non-BGP route that is advertised by BGP is advertised with a LOCAL_PREF value of 100 by default.

To change the local preference metric advertised in the path attribute, include the local-preference statement, specifying a value from 0 through 4,294,967,295 (2^{32} –1):

local-preference *local-preference*;

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Controlling Route Preference

When the JUNOS software determines a route's preference to become the active route, it selects the route with the lowest preference as the active route and installs this route into the forwarding table. By default, the routing software assigns a preference of 170 to routes that originated from BGP. Of all the routing protocols, BGP has the highest default preference value, which means that routes learned by BGP are the least likely to become the active route. (For more information about preferences, see "Route Preferences" on page 6.)

To modify the default BGP preference value, include the preference statement, specifying a value from 0 through 4,294,967,295 (2^{32} – 1):

preference *preference*;

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Examples: Controlling Route Preference

Assign a preference of 160 to routes learned from the BGP system 192.168.1.1. The routing protocol process will prefer these routes over routes learned from other BGP systems, which have the default preference of 170.

```
[edit]
routing-options {
    autonomous-system 23;
}
protocols {
    bgp {
        group 23 {
            type external;
            peer-as 56;
            neighbor 192.168.1.1 {
                preference 160;
            }
        }
    }
}
```

Assign a preference of 140 to all routes learned by BGP systems. Because the default OSPF preference is 150, BGP routes will be preferred over those learned from OSPF.

```
[edit]
routing-options {
    autonomous-system 23;
}
protocols {
    bgp {
        preference 140;
        group 23 {
            type external;
            peer-as 56;
            neighbor 192.168.1.1;
        }
    }
}
```

Configuring Routing Table Path Selection

By default, only the MEDs of routes that have the same peer ASs are compared. You can configure routing table path selection options to get different behaviors. To configure routing table path selection behavior, include the path-selection statement:

```
path-selection {
    (cisco-non-deterministic | always-compare-med | external-router-id);
    med-plus-igp {
        igp-multiplier number;
        med-multiplier number;
    }
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Routing table path selection can be configured in one of the following ways:

- Using the same nondeterministic behavior as does the Cisco IOS software (cisco-non-deterministic). This behavior has two effects:
  - The active path is always first. All nonactive, but eligible, paths follow the active path and are maintained in the order in which they were received, with the most recent path first. Ineligible paths remain at the end of the list.
  - When a new path is added to the routing table, path comparisons are made without removing from consideration those paths that should never be selected because those paths lose the MED tie-breaking rule.

  These two effects cause the system to only sometimes compare the MEDs between paths that it should otherwise compare. Because of this, we recommend that you not configure nondeterministic behavior.

- Always comparing MEDs whether or not the peer ASs of the compared routes are the same (always-compare-med).

  For an example of always comparing MEDs, see "Example: Always Comparing MEDs" on page 570.

- Comparing the router ID between external BGP paths to determine the active path (external-router-id). By default, router ID comparison is not performed if one of the external paths is active.

- Adding the IGP cost to the next-hop destination to the MED before comparing MED values for path selection.

For a description of the algorithm used to determine the active path, see "How the Active Route Is Determined" on page 7.

Example: Always Comparing MEDs

In this example, paths learned from 208.197.169.15 have their MED values compared to the sum of 4 and the MED values of the same paths learned from 208.197.169.14:

```
[edit]
protocols {
    bgp {
        path-selection always-compare-med;
        group ref {
            type external;
            import math;
            peer-as 10458;
            neighbor 208.197.169.14;
        }
        group ref {
            type external;
            peer-as 10;
            neighbor 208.197.169.15;
        }
    }
}

policy-options {
    policy-statement math {
        then {
            metric add 4;
        }
    }
}
```

Configuring BGP to Select Multiple BGP Paths

You can configure BGP to select multiple non-multihop EBGP or IBGP paths as active paths. Selecting multiple paths allows BGP peerings to load-balance traffic across an AS-confederation boundary. The JUNOS BGP multipath supports the following:

- Load-balancing across multiple links between two routers belonging to different ASs
- Load-balancing across a common subnet or multiple subnets to different routers belonging to the same peer AS
- Load-balancing across multiple links between two routers belonging to different external confederation peers
- Load-balancing across a common subnet or multiple subnets to different routers belonging to external confederation peers

To configure a BGP multipath, include the multipath statement:

```
multipath {
    multiple-as;
}
```

To disable the default check requiring that paths accepted by BGP multipath must have the same neighboring AS, include the multiple-as option.

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring a Local AS

You can configure BGP with a different local AS number for each EBGP session, which allows BGP to configure a local AS for each EBGP session. Configuring a local AS simulates a virtual AS for the router. The AS paths for the routes from that EBGP peer have the configured local-as prepended before the peer AS for that session. This is useful if ISP A has acquired another ISP B, but does not want to change the configurations of ISP B's customer routers. ISP B's AS is the AS that is configured as the local AS.

NOTE: If the local AS for the EBGP/IBGP peer is the same as the current AS, do not use the local-as statement to specify the local AS number.

To configure a local AS, include the local-as statement:

```
local-as autonomous-system <private>;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

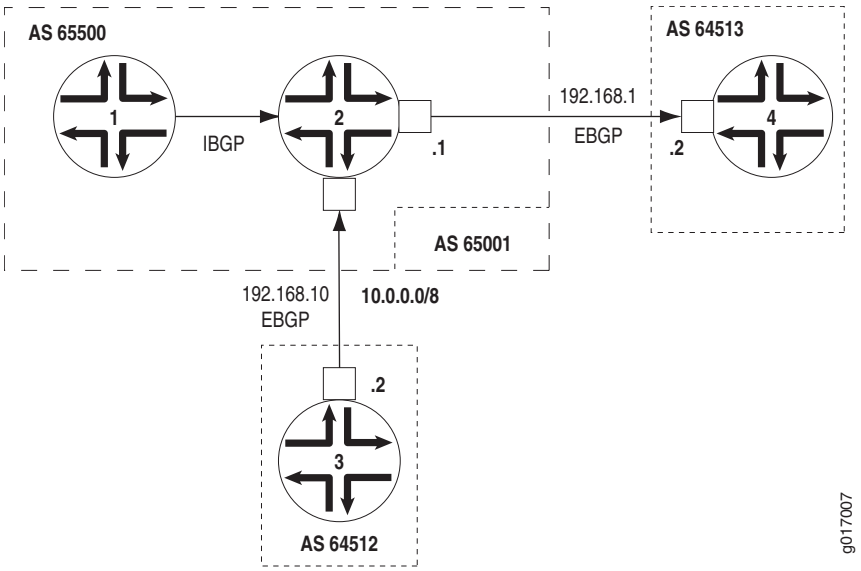
If you include the private keyword, the local AS is not prepended before the peer AS. This means that the AS paths do not show details of such a configuration, and ISP A's EBGP peers and IBGP peers do not see any difference from before the local AS configuration.

Examples: Configuring a Local AS

You can include the local-as statement to configure a router to use a different AS number than the one for which the router is configured. The local AS is used in all BGP protocol exchanges with the routers that are configured for simulating a virtual AS.

Use the local-as statement when ISPs merge and want to preserve a customer's configuration, particularly the AS the customer is configured to peer with. Use the local-as statement to simulate the AS number already in place in customer routers, even if the ISP's router is in a different AS now.

Figure 10: Local AS Configuration



In Figure 10, Router 1 and Router 2 are in AS 65500, Router 4 is in AS 64513, and Router 3 is in AS 64512. Router 2 uses AS 65001 as its local AS.

Router 2 adds AS 65001 when announcing Router 1's routes to Router 3. Router 3 sees an AS path of 65001 65500 64512 for the prefix 10/8. To prevent Router 2 from adding the virtual AS number in its announcements to other peers, use the local-as *autonomous-system* private statement. The local-as *autonomous-system* private statement configures Router 2 to not include the virtual AS number configured in local-as when announcing Router 1's routes to Router 3. In this case, Router 3 sees an AS path of 65500 64512 for the prefix 10/8.

The configuration for each router follows.

On Router 1:

```
routing-options {
   autonomous-system 65500;
}
protocols {
   bgp {
      group internal-AS65500 {
         type internal;
         local-address 10.1.1.1;
         neighbor 10.1.1.2;
      }
   }
}
```

On Router 2:

```
routing-options {
   autonomous-system 65500;
}
protocols {
   bgp {
      group internal-AS65500 {
         type internal;
         local-address 10.1.1.2;
         neighbor 10.1.1.1;
      }
      group external-AS64513 {
         type external;
         peer-as 64513;
         neighbor 192.168.1.2;
      }
      group external-AS64512 {
         type external;
         peer-as 64512;
         neighbor 192.168.10.2;
      }
   }
}
```

On Router 3:

```
routing-options {
   autonomous-system 64512;
}
protocols {
   bgp {
      group external-AS65001 {
         type external;
         peer-as 65001;
         neighbor 192.168.10.1;
      }
   }
}
```

On Router 4:

```
routing-options {
   autonomous-system 64513;
}
protocols {
   bgp {
      group external-65500 {
         peer-as 65500;
         neighbor 192.168.1.1;
      }
   }
}
```

Removing Private AS Numbers from AS Paths

By default, when BGP advertises AS paths to remote systems, it includes all AS numbers, including private AS numbers. You can configure the software so that it removes private AS numbers from AS paths. Doing this is useful when all the following circumstances are true:

- A remote AS for which you provide connectivity is multihomed, but only to the local AS.
- The remote AS does not have an officially allocated AS number.
- It is not appropriate to make the remote AS a confederation member AS of the local AS.

To have the local system strip private AS numbers from the AS path, include the remove-private statement:

```
remove-private;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

CAUTION: Changing configuration statements that affect BGP peerings, such as enabling or disabling remove-private or renaming a BGP group, resets the BGP sessions. Changes that affect BGP peerings should only be made when resetting a BGP session is acceptable.

NOTE: The remote-private statement is applicable only when advertising routers to another neighbor.

The AS numbers are stripped from the AS path starting at the left end of the AS path (the end where AS paths have been most recently added). This operation takes place after any confederation member ASs have already been removed from the AS path, if applicable.

The software is preconfigured with knowledge of the set of AS numbers that is considered private, a range that is defined in the Internet Assigned Numbers Authority (IANA) assigned numbers document. The set of AS numbers reserved as private are in the range from 64,512 through 65,534, inclusive.

Configuring Route Reflection

In standard internal BGP implementations, all BGP systems within the AS are fully meshed so that any external routing information is redistributed among all routers within the AS. This type of implementation can present scaling issues when an AS has a large number of internal BGP systems because of the amount of identical information that BGP systems must share with each other. Route reflection provides one means of decreasing BGP control traffic and minimizing the number of update messages sent within the AS.

In route reflection, BGP systems are arranged in *clusters*. Each cluster consists of at least one system that acts as a *route reflector*, along with any number of *client peers*. BGP peers outside the cluster are called *nonclient peers*. The route reflector reflects (redistributes) routing information to each client peer (*intracluster reflection*) and to all nonclient peers (*intercluster reflection*). Because the route reflector redistributes routes within the cluster, the BGP systems within the cluster do not have to be fully meshed.

When the route reflector receives a route, it selects the best path. Then, if the route came from a nonclient peer, the route reflector sends the route to all client peers within the cluster. If the route came from a client peer, the route reflector sends it to all nonclient peers and to all client peers except the originator. In this process, none of the client peers send routes to other client peers.

To configure route reflection, you specify a cluster identifier only on the BGP systems that are to be the route reflectors. These systems then determine, from the network reachability information they receive, which BGP systems are part of its cluster and are client peers, and which BGP systems are outside the cluster and are nonclient peers.

To configure a router to be a route reflector, you must do the following:

- Configure multiple IBGP groups.
- Configure a cluster identifier (using the cluster statement) for groups that are members of the cluster.
- Configure all the groups with the same IBGP AS number.

To configure the route reflector, include the following statements in the configuration:

```
group group-name {
    type internal;
    peer-as autonomous-system;
    neighbor address1;
    neighbor address2;
}
group group-name {
    type internal;
    peer-as autonomous-system;
    cluster cluster-identifier;
    neighbor address3;
    neighbor address4;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

By default, the BGP route reflector performs intracluster reflection because it assumes that all the client peers are not fully meshed. However, if the client peers are fully meshed, intracluster reflection results in the sending of redundant route advertisements. In this case, you can disable intracluster reflection by including the no-client-reflect statement within the group statement:

```
group group-name {
    type internal;
    peer-as autonomous-system;
    cluster cluster-identifier;
    no-client-reflect;
    neighbor address3;
    neighbor address4;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement

NOTE: BGP route reflection is not supported for VPN routing and forwarding (VRF) routing instances.

Examples: Configuring Route Reflection

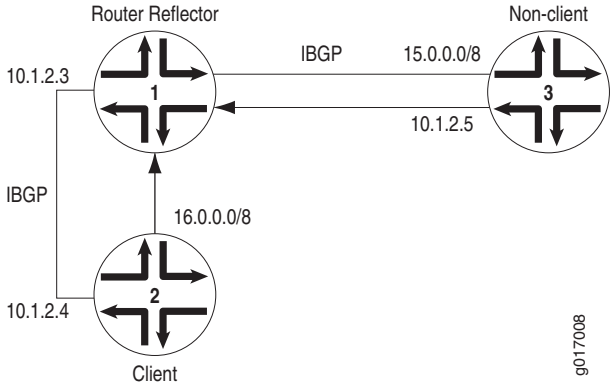
This example shows how to configure a simple route reflector. The configuration shown in Figure 9 contains three routes: Router 1, which is the route reflector; Router 2, which is a client; and Router 3, which is a nonclient.

The routers have the following loopback addresses:

- Router 1—10.1.2.3
- Router 2—10.1.2.4
- Router 3—10.1.2.5

You must configure all routers to run a common IGP or to have static configuration, so that they learn each other's loopback addresses.

Figure 11: Simple Route Reflector



Configure Router 1 to be a route reflector for Router 2 and a regular IBGP neighbor for Router 3:

```
[edit]
routing-options {
    autonomous-system 65534;
}
protocols {
    bgp {
        group 13 {
            type internal;
            local-address 10.1.2.3;
            neighbor 10.1.2.5;
        }
        group 12 {
            type internal;
            local-address 10.1.2.3;
            cluster 1.2.3.4;
            neighbor 10.1.2.4;
        }
    }
}
```

Configure Router 2 to be an IBGP neighbor to Router 1 and announce 16.0.0.0/8 to Router 1. Configure route 16.0.0.0/8 as a static route on Router 2.

```
[edit]
routing-options {
    static {
        route 16.0.0.0/8 nexthop 172.16.1.2;
    }
    autonomous-system 65534;
}
protocols {
    bgp {
        group 21 {
            type internal;
            local-address 10.1.2.4;
            export dist-static;
            neighbor 10.1.2.3;
        }
    }
}
policy-options {
    policy-statement dist-static {
        from protocol static;
        then accept;
    }
}
```

Configure Router 3 to be an IBGP neighbor to Router 1 and announce 15.0.0.0/8 to Router 1. Configure route 15.0.0.0/8 as a static route on Router 3.

```
[edit]
routing-options {
    static {
        route 15.0.0.0/8 nexthop 172.16.1.2;
    }
    autonomous-system 65534;
}
```

```
protocols {
    bgp {
        group 31 {
        type internal;
        local-address 10.1.2.5;
        export dist-static;
        neighbor 10.1.2.3;
        }
    }
}
policy-options {
    policy-statement dist-static {
        from protocol static;
        then accept;
    }
}
```

The following is the output of the show route detail command for route 16.0.0.0/8 on Router 1 and Router 3. Note that router 1 learns 16.0.0.0/8 from its client, Router 2, and reflects it to Router 3. On Router 3, the output of the show route commands include the cluster list and originator ID attributes, which are added by Router 1 when the route is reflected.

Router 1:

```
user@router1> show route 16.0.0.0/8 detail

inet.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

16.0.0.0/8 (1 entry, 1 announced)
    *BGP        Preference: 170/-101
                Source: 10.1.2.4
                Nexthop: 172.16.1.2 via fxp0.0, selected
                State: <Active Int Ext>
                Local AS: 65534                 Peer AS: 65534
                Age: 11:55                      Metric2: 0
                Task: BGP_65534.10.1.2.4+4327
                Announcement bits (3): 2-KRT 3-BGP.0.0.0.0+179 4-BGP_Sync_Any
                AS path: I
                BGP next hop: 172.16.1.2
                Localpref: 100
                Router ID: 10.1.2.4
```

Router 3:

```
user@router3> show route 16.0.0.0/8 detail

inet.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

16.0.0.0/8 (1 entry, 1 announced)
    *BGP        Preference: 170/-101
                Source: 10.1.2.3
                Nexthop: 172.16.1.2 via fxp0.0, selected
                State: <Active Int Ext>
                Local AS: 65534                 Peer AS: 65534
                Age: 11:57              Metric2: 0
                Task: BGP_65534.10.1.2.3+4619
                Announcement bits (2): 2-KRT 4-BGP_Sync_Any
                AS path: I <Originator>
                Cluster list: 1.2.3.4
```

```
                Originator ID: 10.1.2.4
                BGP next hop: 172.16.1.2
                Localpref: 100
                Router ID: 10.1.2.3
```

The following is the output of the show route detail command for route 15.0.0.0/8 on router 1 and router 2. Similar to when routes are reflected from client peers to nonclient peers, router 1 reflects a route it learns from a regular IBGP neighbor to its client. Cluster list and Originator ID attributes are added during the reflection process.

Router 1:

```
user@router1> show route 15.0.0.0/8 detail

inet.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

15.0.0.0/8 (1 entry, 1 announced)
    *BGP        Preference: 170/-101
                Source: 10.1.2.5
                Nexthop: 172.16.1.2 via fxp0.0, selected
                State: <Active Int Ext>
                Local AS: 65534                 Peer AS: 65534
                Age: 11:14                      Metric2: 0
                Task: BGP_65534.10.1.2.5+179
                Announcement bits (3): 2-KRT 3-BGP.0.0.0.0+179 4-BGP_Sync_Any
                AS path: I
                BGP next hop: 172.16.1.2
                Localpref: 100
                Router ID: 10.1.2.5
```

Router 2:

```
user@router2> show route 15.0.0.0/8 detail

inet.0: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

15.0.0.0/8 (1 entry, 1 announced)
    *BGP        Preference: 170/-101
                Source: 10.1.2.3
                Nexthop: 172.16.1.2 via fxp0.0, selected
                State: <Active Int Ext>
                Local AS: 65534                 Peer AS: 65534
                Age: 11:23                      Metric2: 0
                Task: BGP_65534.10.1.2.3+179
                Announcement bits (2): 2-KRT 4-BGP_Sync_Any
                AS path: I <Originator>
                Cluster list: 1.2.3.4
                Originator ID: 10.1.2.5
                BGP next hop: 172.16.1.2
                Localpref: 100
                Router ID: 10.1.2.3
```

Enabling Route Flap Damping

BGP *route flapping* describes the situation in which BGP systems send an excessive number of update messages to advertise network reachability information. BGP *flap damping* is a method of reducing the number of update messages sent between BGP peers, thereby reducing the load on these peers, without adversely affecting the route convergence time for stable routes.

By default, route flap damping is disabled. To enable it, include the damping statement:

```
damping;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Damping is applied to external peers and to peers at confederation boundaries. For finer control over which peers have damping enabled, include the damping statement at the [edit protocols bgp group *group-name*] level.

By default, route flap damping uses the following parameters:

- Decay half-life while reachable—15 minutes
- Maximum hold-down time—60 minutes
- Reuse threshold—750
- Cutoff threshold—3000

To change these default parameters, you must define the flap damping parameters with the damping statement at the [edit policy-options] hierarchy level and then apply them using an import statement when configuring BGP. For more information about flap damping and defining flap damping parameters, see the *JUNOS Policy Framework Configuration Guide*. For more information about applying policy filters in BGP, see “Configuring BGP Routing Policy” on page 594.

Enabling Multiprotocol BGP

Multiprotocol BGP (MBGP) is an extension to BGP that enables BGP to carry routing information for multiple network layers and address families. MBGP can carry the unicast routes used for multicast routing separately from the routes used for unicast IP forwarding.

To enable MBGP, you configure BGP to carry network layer reachability information (NLRI) for address families other than unicast IPv4 by including the family inet statement:

```
family inet {
    (any | labeled-unicast | multicast | unicast) {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
        }
        rib-group group-name;
    }
}
```

To enable MBGP to carry NLRI for the IPv6 address family, include the family inet6 statement:

```
family inet6 {
    (any | labeled-unicast | multicast | unicast) {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
    }
}
```

To enable MBGP to carry Layer 3 VPN NLRI for the IPv4 address family, include the family inet-vpn statement:

```
family inet-vpn {
    (any | multicast | unicast) {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
    }
}
```

To enable MBGP to carry Layer 3 VPN NLRI for the IPv6 address family, include the family inet6-vpn statement:

```
family inet6-vpn {
    (any | multicast | unicast) {
        prefix-limit {
            maximum number;
            teardown <percentage> <idle-timeout (forever | minutes)>;
        }
        rib-group group-name;
    }
}
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

NOTE: If you change the address family specified in the [edit bgp family inet] hierarchy level, the BGP sessions are dropped and then reestablished.

By default, BGP peers carry only unicast routes used for unicast forwarding purposes. To configure BGP peers to carry only multicast routes, specify the multicast option. To configure BGP peers to carry both unicast and multicast routes, specify the any option.

When MBGP is configured, BGP installs the MBGP routes into different routing tables. Each routing table is identified by the protocol family or address family indicator (AFI) and a subaddress family indicator (SAFI).

The JUNOS software supports all unicast and multicast SAFIs (1 and 2) for both AFI 1 (IPv4) and AFI 2 (IPv6). The following table shows all possible AFI/SAFI combinations and routing tables populated with this information:

	SAFI 1	SAFI 2
AFI 1 (IPv4)	inet.0	inet.2
AFI 2 (IPv6)	inet6.0	inet6.2

If peers are not MBGP, you cannot export routes from inet.2 to them, only routes in the inet.0 routing table. Routes in inet.2 can be sent only to MBGP peers, since they are sent with subaddress family information that identifies them as routes to multicast sources. The inet.2 table should be a subset of the routes that you have in inet.0, since it is unlikely that you would have a route to a multicast source to which you could not send unicast traffic.

The inet.2 routing table is used to keep the unicast routes that are used for multicast reverse-path-forwarding checks. You automatically get an inet.2 routing table when you configure MBGP (by setting NLRI to any). The additional reachability information learned by MBGP from the NLRI multicast updates are placed in inet.2.

When you enable multiprotocol BGP, you can do the following:

- Limiting the Number of Prefixes on a BGP Peering on page 583
- Configuring BGP Routing Table Groups on page 584
- Resolving Routes to PE Routers Located in Other ASs on page 584
- Allowing Labeled and Unlabeled Routes on page 585

Limiting the Number of Prefixes on a BGP Peering

You can limit the number of prefixes received on a BGP peering, and log rate-limited messages when the number of injected prefixes exceeds a set limit. You can also tear down the peering when the number of prefixes exceeds the limit.

To configure a maximum number of prefixes, include the prefix-limit statement:

```
prefix-limit {
    maximum number;
    teardown <percentage> <idle-timeout (forever | minutes)>;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

When you set the maximum number of prefixes, a message is logged when that number is reached.

If you include the teardown statement, the session is torn down when the maximum number of prefixes is reached. If you specify a percentage, messages are logged when the number of prefixes reaches that percentage. Once the session is torn down, it is reestablished in a short time (unless you include the idle-timeout statement). Then the session can be kept down for a specified amount of time, or forever. If you specify forever, the session is reestablished only after the you use a clear bgp neighbor command.

Configuring BGP Routing Table Groups

When a BGP session receives a unicast or multicast NLRI, it installs the route in the appropriate table (inet.0 or inet6.0 for unicast, and inet.2 or inet6.2 for multicast). To add unicast prefixes to both the unicast and multicast tables, you can configure BGP routing table groups. This is useful if you cannot perform multicast NLRI negotiation.

To configure BGP routing table groups, include the rib-group statement:

```
rib-group group-name;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Resolving Routes to PE Routers Located in Other ASs

You can allow labeled routes to be placed in the inet.3 routing table for route resolution. These routes are then resolved for provider edge (PE) router connections where the remote PE is located across another AS. For a PE router to install a route in the VRF routing instance, the next hop must resolve to a route stored within the inet.3 table.

To resolve routes into the inet.3 routing table, include the resolve-vpn statement:

```
resolve-vpn group-name;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Allowing Labeled and Unlabeled Routes

You can allow both labeled and unlabeled routes to be exchanged in a single session. The labeled routes are placed in the inet.3 routing table, and both labeled and unlabeled unicast routes can be sent or received by the router.

To allow both labeled and unlabeled routes to be exchanged, include the rib inet.3 statement:

```
rib inet.3;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Enabling BGP to Carry Flow-Specific Routes

You can allow BGP to carry flow-specific NLRI messages. Flow routes are encapsulated into the flow-specific NLRI and propagated through a network or VPNs, sharing filter-like information. Flow routes are an aggregation of match conditions and resulting actions for packets. They provide you with traffic filtering and rate-limiting capabilities much like firewall filters.

When you enable flow-specific routes, you can do the following:

- Configuring Flow-Specific Routes for IPv4 Unicast on page 585
- Configuring Flow-Specific Routes for Layer 3 VPNs on page 586

Configuring Flow-Specific Routes for IPv4 Unicast

To enable MBGP to carry flow-specific NLRI for the inet address family, include the flow statement:

```
flow;
```

NOTE: Unicast flow routes are supported for the default instance, VRF instances, and virtual-router instances only. Instance type is configured with the instance-type statement at the [edit routing-instance *instance-name*] hierarchy level.

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Flow routes received using the BGP NLRI messages are validated before they are installed into the flow routing table *instance-name*.inetflow.0. The validation procedure is described in the Internet draft *Dissemination of Flow Specification Rules,* draft-marques-idr-flow-spec-02.txt. You can bypass the validation process and use your own specific import policy.

To disable the validation procedure and use an import policy instead, include the no-validate statement at the [edit protocols bgp group *group-name* family inet flow] hierarchy level:

```
no-validate policy-name;
```

Configuring Flow-Specific Routes for Layer 3 VPNs

The VPN compares the route target extended community in the NLRI to the import policy. If there is a match, the VPN can start using the flow routes to filter and rate-limit packet traffic. Received flow routes are installed into the flow routing table *instance-name*.inetflow.0.

Flow routes can also be propagated throughout a VPN network and shared among VPNs, providing filter and rate-limiting capabilities.

To enable MBGP to carry flow-specific NLRI for the inet-vpn address family, include the flow statement at the [edit protocols bgp group *group-name* family inet-vpn] hierarchy level:

```
flow;
```

NOTE: VPN flow routes are supported for the default instance only. Instance type is configured with the instance-type statement at the [edit routing-instance *instance-name*] hierarchy level.

Flow routes configured for VPNs with family inet-vpn are not automatically validated, so the no-validate statement is not supported at the [edit protocols bgp group *group-name* family inet-vpn] hierarchy level.

For more information on flow routes, see "Configuring a Flow Route" on page 96 and the Internet draft *Dissemination of Flow Specification Rules,* draft-marques-idr-flow-spec-02.txt.

Enabling BGP to Carry Connectionless Network Services Routes

Connectionless Network Services (CLNS) is a Layer 3 protocol similar to Internet Protocol version 4 (IPv4). CLNS uses network service access points (NSAPs) to address end systems. This allows for a seamless AS based on ISO NSAPs.

NOTE: CLNS is supported for the J-series Services Router only.

A single routing domain consisting of ISO NSAP devices are considered to be CLNS islands. CLNS islands are connected together by VPNs.

You can configure BGP to exchange ISO CLNS routes between PE routers connecting various CLNS islands in a VPN using multiprotocol BGP extensions. These extensions are the ISO VPN NLRIs.

To enable MBGP to carry CLNS VPN NLRIs, include the iso-vpn statement:

```
iso-vpn {
    unicast {
        prefix-limit number;
        rib-group group-name;
    }
}
```

To limit the number of prefixes from a peer, include the prefix-limit statement. To specify a routing table group, include the rib-group statement.

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Each CLNS network island is treated as a separate VRF instance on the PE router.

You can configure CLNS on the global level, group level, and neighbor level.

For information on CLNS, see "Configuring Support for Connectionless Network Services" on page 278 and the *J-series Services Router Advanced WAN Access Configuration Guide*.

Example: Enabling CLNS Between Two Routers

Configure CLNS between two routers through a route reflector:

On Router 1:

```
[edit protocols bgp]
protocols {
    bgp {
        local-address 10.255.245.195;
        group pe-pe {
            type internal;
            neighbor 10.255.245.194 {
                family iso-vpn {
                    unicast;
                }
            }
        }
    }
}

[edit routing-instances]
routing-instances {
    aaaa {
        instance-type vrf;
        interface fe-0/0/0.0;
        interface so-1/1/0.0;
        interface lo0.1;
        route-distinguisher 10.255.245.194:1;
        vrf-target target:11111:1;
        protocols {
            isis {
                export dist-bgp;
                no-ipv4-routing;
                no-ipv6-routing;
                clns-routing;
                interface all;
            }
        }
    }
}
```

On Router 2:

```
[edit protocols bgp]
protocols {
    bgp {
        group pe-pe {
            type internal;
            local-address 10.255.245.198;
            family route-target;
            neighbor 10.255.245.194 {
                family iso-vpn {
                    unicast;
                }
            }
        }
    }
}

[edit routing-instances]
routing-instances {
    aaaa {
        instance-type vrf;
        interface lo0.1;
        interface so-0/1/2.0;
        interface so-0/1/3.0;
        route-distinguisher 10.255.245.194:1;
        vrf-target target:11111:1;
        routing-options {
            rib aaaa.iso.0 {
                static {
                    iso-route 47.0005.80ff.f800.0000.bbbb.1022/104 next-hop
                      47.0005.80ff.f800.0000.aaaa.1000.1921.6800.4196.00;
                }
            }
        }
        protocols {
            isis {
                export dist-bgp;
                no-ipv4-routing;
                no-ipv6-routing;
                clns-routing;
                interface all;
            }
        }
    }
}
```

On Route Reflector:

```
[edit protocols bgp]
protocols {
    bgp {
        group pe-pe {
            type internal;
            local-address 10.255.245.194;
            family route-target;
            neighbor 10.255.245.195 {
                cluster 0.0.0.1;
            }
            neighbor 10.255.245.198 {
                cluster 0.0.0.1;
            }
        }
    }
}
```

Example: Configuring CLNS Within a VPN

Configure CLNS on three PE routers within a VPN:

On PE Router 1:

```
[edit protocols bgp]
protocols {
    mpls {
        interface all;
    }
    bgp {
        group asbr {
            type external;
            local-address 10.245.245.3;
            neighbor 10.245.245.1 {
                multihop;
                family iso-vpn {
                    unicast;
                }
                peer-as 200;
            }
        }
    }
}
```

```
[edit routing-instances]
routing-instances {
    aaaa {
        instance-type vrf;
        interface lo0.1;
        interface t1-3/0/0.0;
        interface fe-5/0/1.0;
        route-distinguisher 10.245.245.1:1;
        vrf-target target:11111:1;
        protocols {
            isis {
                export dist-bgp;
                no-ipv4-routing;
                no-ipv6-routing;
                clns-routing;
                interface all;
            }
        }
    }
}
```

On PE Router 2:

```
[edit protocols bgp]
protocols {
    bgp {
        group asbr {
            type external;
            multihop;
            local-address 10.245.245.1;
            family iso-vpn {
                unicast;
            }
            neighbor 10.245.245.2 {
                peer-as 300;
            }
            neighbor 10.245.245.3 {
                peer-as 100;
            }
        }
    }
}

[edit routing-instances]
routing-instances {
    aaaa {
        instance-type vrf;
        interface lo0.1;
        route-distinguisher 10.245.245.1:1;
        vrf-target target:11111:1;
    }
}
```

On PE Router 3:

```
[edit protocols bgp]
protocols {
    bgp {
        group asbr {
            type external;
            multihop;
            local-address 10.245.245.2;
            neighbor 10.245.245.1 {
                family iso-vpn {
                    unicast;
                }
                peer-as 200;
            }
        }
    }
}

[edit routing-instances]
routing-instances {
    aaaa {
        instance-type vrf;
        interface lo0.1;
        interface fe-0/0/1.0;
        interface t1-3/0/0.0;
        route-distinguisher 10.245.245.1:1;
        vrf-target target:11111:1;
        protocols {
            isis {
                export dist-bgp;
                no-ipv6-routing;
                clns-routing;
                interface all;
            }
        }
    }
}
```

Enabling Route Target Filtering

You can limit the number of prefixes advertised on BGP peerings specifically to the peers that need the updates.

In a VPN provider network, a BGP speaker advertises all VPN routes to the peers in the same VPN. Peers that are configured either as a route reflector or border router for a VPN must store all routes within the network. While PE routers automatically discard routes that do not affect them, these route updates must still be generated and received.

Enabling route target filtering allows you to limit these route updates.

To enable route target filtering, include the route-target statement:

```
route-target {
    advertise-default;
    external-paths number;
    prefix-limit {
        maximum number;
        teardown <percentage> <idle-timeout (forever | time-in-minutes)>;
    }
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

If you include the advertise-default statement, the router advertises the default route-target route (0:0:0/0) and suppresses any specific route-target routes. This is useful for a route reflector in a BGP group consisting of neighbor PE routers only. If you include the external-paths statement, the router limits the number of external paths accepted for route filtering. The range is from 1 through 16. The default is 1. If you include the teardown statement, the session is torn down when the maximum number of prefixes is reached. If you specify a percentage, messages are logged when the number of prefixes reaches that percentage. Once the session is torn down, it is reestablished in a short time. Include the idle-timeout statement to keep the session down for a specified amount of time, or forever. If you specify forever, the session is reestablished only after you use the clear bgp neighbor command.

For more information about VPNs, see the *JUNOS VPNs Configuration Guide*.

Enabling Layer 2 VPN and VPLS Signaling

You can enable BGP to carry Layer 2 VPN and VPLS NLRI messages.

To enable VPN and VPLS signaling, include the **family** statement:

```
family {
    l2vpn {
        signaling {
            prefix-limit {
                maximum number;
                teardown <percentage> <idle-timeout (forever | minutes)>;
            }
        }
    }
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

To configure a maximum number of prefixes, include the **prefix-limit** statement:

```
prefix-limit {
    maximum number;
    teardown <percentage> <idle-timeout (forever | minutes)>;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

When you set the maximum number of prefixes, a message is logged when that number is reached. If you include the **teardown** statement, the session is torn down when the maximum number of prefixes is reached. If you specify a percentage, messages are logged when the number of prefixes reaches that percentage. Once the session is torn down, it is reestablished in a short time. Include the **idle-timeout** statement to keep the session down for a specified amount of time, or forever. If you specify **forever**, the session is reestablished only after you use the **clear bgp neighbor** command.

For more information about VPNs, see the *JUNOS VPNs Configuration Guide*. For a detailed VPLS example configuration, see the *JUNOS Feature Guide*.

Configuring BGP Routing Policy

All routing protocols use the JUNOS software routing table to store the routes that they learn and to determine which routes they should advertise in their protocol packets. Routing policy allows you to control which routes the routing protocols store in and retrieve from the routing table. For information about routing policy, see the *JUNOS Policy Framework Configuration Guide*.

When configuring BGP routing policy, you can perform the following tasks:

Applying Routing Policy

You define routing policy at the [edit policy-options] hierarchy level. To apply policies you have defined for BGP, include the import and export statements within the BGP configuration. For information about defining policy, see the *JUNOS Policy Framework Configuration Guide*.

You can apply policies as follows:

- BGP global import and export statements—Include these statements at the [edit protocols bgp] hierarchy level (for routing instances, include these statements at the [edit routing-instances routing-instance-name protocols bgp] hierarchy level).
- Group import and export statements—Include these statements at the [edit protocols bgp group *group-name*] hierarchy level (for routing instances, include these statements at the [edit routing-instances *routing-instance-name* protocols bgp group *group-name*] hierarchy level).
- Peer import statements—Include these statements at the [edit protocols bgp group *group-name* neighbor *address*] hierarchy level (for routing instances, include these statements at the [edit routing-instances *routing-instance-name* protocols bgp group *group-name* neighbor *address*] hierarchy level). (A peer uses its group's export statement.)

A peer-level import statement overrides a group import statement. A group-level import or export statement overrides a global BGP import or export statement.

To apply policies, see the following sections:

Applying Policies to Routes Being Imported into the Routing Table from BGP

To apply policy to routes being imported into the routing table from BGP, include the import statement, listing the names of one or more policies to be evaluated:

```
import [ policy-names ];
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

If you specify more than one policy, they are evaluated in the order specified, from first to last, and the first matching filter is applied to the route. If no match is found, BGP places into the routing table only those routes that were learned from BGP routers.

Applying Policies to Routes Being Exported from the Routing Table into BGP

To apply policy to routes being exported from the routing table into BGP, include the export statement, listing the names of one or more policies to be evaluated:

```
export [ policy-names ];
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

If you specify more than one policy, they are evaluated in the order specified, from first to last, and the first matching filter is applied to the route. If no routes match the filters, the routing table exports into BGP only the routes that it learned from BGP.

Setting BGP to Advertise Inactive Routes

By default, BGP stores the route information it receives from update messages in the JUNOS routing table, and the routing table exports only active routes into BGP, which BGP then advertises to its peers. To have the routing table export to BGP the best route learned by BGP even if the JUNOS software did not select it to be an active route, include the advertise-inactive statement:

```
advertise-inactive;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring How Often BGP Exchanges Routes with the Routing Table

BGP stores the route information it receives from update messages in the routing table, and the routing table exports active routes from the routing table into BGP. BGP then advertises the exported routes to its peers. By default, the exchange of route information between BGP and the routing table occurs immediately after the routes are received. This immediate exchange of route information might cause instabilities in the network reachability information. To guard against this, you can delay the time between when BGP and the routing table exchange route information.

To configure how often BGP and the routing table exchange route information, include the out-delay statement:

```
out-delay seconds;
```

By default, the routing table retains some of the route information learned from BGP. To have the routing table retain all or none of this information, include the keep statement:

```
keep (all | none);
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

The routing table can retain the route information learned from BGP in one of the following ways:

- Default (omit the keep statement)—Keep all route information that was learned from BGP except for routes whose AS path is looped and the loop includes the local AS.
- keep all—Keep all route information that was learned from BGP.
- keep none—Discard routes that were received from a peer and that were rejected by import policy or other sanity checking, such as AS path or next hop. When you configure keep none for the BGP session and the inbound policy changes, the JUNOS software forces readvertisement of the full set of routes advertised by the peer.

In an AS path healing situation, routes with looped paths theoretically could become usable during a soft reconfiguration when the AS path loop limit is changed. However, there is a significant memory usage difference between the default and keep all because it is common for a peer to readvertise routes back to the peer from which it learned them. The default behavior is not to waste memory on such routes.

Disabling Suppression of Route Advertisements

The JUNOS software does not advertise the routes learned from one external BGP (EBGP) peer back to the same EBGP peer. In addition, the software does not advertise those routes back to any EBGP peers that are in the same AS as the originating peer, regardless of the routing instance. You can modify this behavior by including the advertise-peer-as statement in the configuration. To disable the default advertisement suppression, including the advertise-peer-as statement:

```
advertise-peer-as;
```

NOTE: The route suppression default behavior is disabled if the as-override statement is included in the configuration.

If you include the advertise-peer-as statement in the configuration, BGP advertises the route regardless of this check.

To restore the default behavior, include the no-advertise-peer-as statement in the configuration.:

```
no-advertise-peer-as;
```

If you include both the as-override and no-advertise-peer-as statements in the configuration, the no-advertise-peer-as statement is ignored. You can include these statements at multiple hierarchy levels.

For a list of hierarchy levels at which you can configure these statements, see the statement summary section for this statement.

Configuring EBGP Peering Using IPv6 Link-local Address

The JUNOS software supports EBGP peering sessions by means of IPv6 link-local addresses. An IPv6 peering session can be configured when a 128-bit IPv6 address is specified in the neighbor statement. The peer address is identified as link-local by means of the local-interface statement.

To configure an EBGP peer, specify a 128-bit IPv6 link-local address in the neighbor statement:

```
neighbor ipv6-link-local-address;
```

To specify the interface name for the EBGP link-local peer, include the local-interface statement:

```
local-interface interface-name;
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for this statements.

This statement is valid only for 128-bit IPv6 link-local addresses and is mandatory for configuring an IPv6 EBGP link-local peering session. For more information about IPv6 addressing, see “Routing Protocols Concepts” on page 3.

NOTE: Configuring EBGP peering using link-local addresses is only applicable for directly connected interfaces. There is no support for multihop peering.

Configuring IPv6 BGP Routes over IPv4 Transport

You can export both IPv6 and IPv4 prefixes over an IPv4 connection where both sides are configured with an IPv4 interface. In this case, the BGP neighbors are IPv4 prefixes. The IPv4-compatible IPv6 prefixes are configured on the interfaces to preclude the configuration of static routes.

Keep the following in mind when exporting IPv6 BGP prefixes:

- BGP derives next-hop prefixes using the IPv4-compatible IPv6 prefix. For example, the IPv4 next-hop prefix 10.19.1.1 translates to the IPv6 next-hop prefix ::10.19.1.1 (hexadecimal format ::a13:101).

NOTE: There must be an active route to the IPv4-compatible IPv6 next hop to export IPv6 BGP prefixes.

- An IPv6 connection must be configured over the link. The connection must be either an IPv6 tunnel or a dual-stack configuration.
- When configuring IPv4-compatible IPv6 prefixes, use a mask that is longer than 96 bits.
- Configure a static route if you want to use normal IPv6 prefixes.

Example: Configuring IPv6 BGP Routes over IPv4 Transport

Configure IPv4 transport from interface ge-0/1/0 with an IPv4 prefix 11.19.1.2/24 to interface ge-1/1/1 with an IPv4 prefix 11.19.1.1/24 to carry IPv6 BGP routes.

Define IPv4 and IPv6 BGP groups for 11.19.1.2 with BGP neighbor 11.19.1.1:

```
[edit protocols]
bgp {
    group ebgp_both {
        type external;
        local-address 11.19.1.2;
        family inet {
            unicast;
        }
        family inet6 {
            unicast;
        }
        peer-as 1;
        neighbor 11.19.1.1;
    }
}
```

Configure the interfaces with both an IPv4 and a corresponding IPv4-compatible IPv6 prefix:

```
[edit interfaces]
ge-0/1/0 {
    unit 0 {
        family inet {
            address 11.19.1.2/24;
        }
        family inet6 {
            address ::11.19.1.2/126;
        }
    }
}
```

Configuring BGP to Log System Log Messages

Whenever a BGP peer makes a state transition, you can configure BGP so that it generates a syslog message. To do this, include the log-updown statement:

```
log-updown;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

NOTE: Enabling the log-updown statement causes BGP state transitions to be logged at warning level.

Describing BGP Router Configuration

You can enter plain text to describe the BGP router configuration.

To enter a description, include the description statement:

```
description description-text;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Blocking Non-Peer TCP Connection Attempts

You can restrict Transmission Control Protocol (TCP) connection attempts on port 179 to BGP peers only. This blocks all non-BGP connection attempts on port 179.

To restrict TCP connection attempts to BGP peers include the apply-path statement:

```
apply-path protocol bgp group group-name neighbor neighbor;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

For detailed information about configuring TCP connection attempts, see the *JUNOS Policy Framework Configuration Guide*.

Applying BGP Export Policy to VRF Routes

You can apply a VPN routing and forwarding (VRF) export policy in addition to applying a BGP export policy to routes before advertising the routes to provider edge (PE) routers in a VPN. The default action is to accept routes.

To apply an export policy to routes, include the vpn-apply-export statement:

```
vpn-apply-export;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Enabling Next-Hop Reachability Information

To enable multiprotocol updates to contain next-hop reachability information, include the include-mp-next-hop statement:

```
include-mp-next-hop;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Configuring the BFD Protocol

The bidirectional forwarding detection (BFD) protocol is a simple hello mechanism that detects failures in a network. BFD works with a wide variety of network environments and topologies. The failure detection timers for BFD have shorter time limits than default failure detection mechanisms, providing faster detection. These timers are also adaptive and can be adjusted to be more or less aggressive.

NOTE: BFD is supported only by single-hop EBGP sessions.

To enable failure detection, include the bfd-liveness-detection statement:

```
bfd-liveness-detection {
    detection-time {
        threshold milliseconds;
    }
    minimum-interval milliseconds;
    minimum-receive-interval milliseconds;
    transmit-interval {
        threshold milliseconds;
        minimum-interval milliseconds;
    }
    multiplier number;
    version (0 | 1 | automatic);
}
```

To specify the threshold for the adaptation of the detection time, include the threshold statement:

```
detection-time {
    threshold milliseconds;
}
```

To specify the minimum transmit and receive interval for failure detection, include the minimum-interval statement:

```
minimum-interval milliseconds;
```

NOTE: Specifying an interval smaller than 300ms can cause undesired BFD flapping.

To specify only the minimum receive interval for failure detection, include the minimum-receive-interval statement:

```
minimum-receive-interval milliseconds;
```

To specify the detection time multiplier for failure detection, include the multiplier statement:

```
multiplier number;
```

To specify the threshold for detecting the adaptation of the transmit interval, include the threshold statement:

```
transmit-interval {
    threshold milliseconds;
}
```

The threshold value must be greater than the transmit interval.

To specify only the minimum transmit interval for failure detection, include the minimum-interval statement:

```
transmit-interval {
   minimum-interval milliseconds;
}
```

To specify the BFD version used for detection, include the version statement:

```
version (1 | automatic);
```

For a list of hierarchy levels at which you can configure these statements, see the statement summary sections for these statements.

Configuring the Segment Size for TCP

TCP path MTU discovery helps avoid BGP packet fragmentation. However, enabling TCP path MTU discovery creates ICMP vulnerability. To prevent these ICMP vulnerability issues, you can configure the TCP maximum segment size (MSS) globally, or for each BGP peer. You can also configure the advertised MSS value for each BGP peer to prevent fragmentation of packets sent by the BGP peer.

To configure the TCP MSS value, include the tcp-mss statement:

```
tcp-mss segment-size;
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

Include the tcp-mss statement for a specific BGP neighbor to send the specified segment size to the BGP neighbor as the advertised MSS. The configured MSS value is also used as the maximum segment size for the sender. If the MSS value from the BGP neighbor is less than the MSS value configured, the MSS value from the BGP neighbor is used as the maximum segment size for the sender.

Tracing BGP Protocol Traffic

To trace BGP protocol traffic, you can specify options in the global traceoptions statement at the [edit routing-options] hierarchy level, and you can specify BGP-specific options by including the traceoptions statement at the [edit protocols bgp] hierarchy level. For routing instances, include the statement.

```
traceoptions {
    file name <replace> <size size> <files number> <no-stamp>
      <(world-readable | no-world-readable)>;
    flag flag <flag-modifier> <disable>;
}
```

For a list of hierarchy levels at which you can configure this statement, see the statement summary section for this statement.

You can specify the following BGP-specific options in the BGP traceoptions statement:

- aspath—Trace AS path regular expression operations.
- damping—Trace damping operations.
- keepalive—Trace BGP keepalive messages.
- open—Trace BGP open packets. These packets are sent between peers when they are establishing a connection.
- packets—Trace all BGP protocol packets.
- update—Trace update packets. These packets provide routing updates to BGP systems.

You can filter trace statements and output only the statement information that passes through the filter by specifying the filter flag modifier. The filter modifier is only supported for the route and damping tracing flags.

NOTE: Per-neighbor traceoption filtering is not supported on a BGP per-neighbor level for route and damping flags. Trace option filtering support is on a peer group level.

NOTE: Use the traceoption flags detail and all with caution. These flags may cause the CPU to become very busy.

The match-on statement specifies filter matches based on prefixes. It is used to match on route filters.

For general information about tracing, see the tracing and logging information in the *JUNOS System Basics Configuration Guide*.

Examples: Tracing BGP Protocol Traffic

Trace only unusual or abnormal operations to routing-log, and trace detailed information about all BGP messages to bgp-log:

```
[edit]
routing-options {
    traceoptions {
        file routing-log;
    }
    autonomous-system 23;
}
protocols {
    bgp {
        group 23 {
            type external;
            peer-as 56;
            traceoptions {
                file bgp-log size 10k files 5;
                flag packets detail;
            }
            0.0.0.0/0;
        }
    }
}
```

Trace only update messages received from the configured peer:

```
[edit]
routing-options {
    autonomous-system 23;
    router-id 10.0.0.1;
}
protocols {
    bgp {
      group 23 {
        type external;
        peer-as 56;
        neighbor boojum.snark.net {
          traceoptions {
            file bgp-log size 10k files 2;
            flag update detail;
          }
        }
      }
    }
}
```

Trace only messages that pass the policy based on prefix match:

```
[edit]
protocols {
    bgp {
      traceoptions {
        file bgp-tr size 5m files 10;
        flag route filter policy couple-route match-on prefix;
      }
    }
}
```