Chapter 14
Border Gateway Protocol (BGP)

14.1 BGP Overview

The Border Gateway Protocol (BGP) is an exterior, or inter-domain, routing protocol used for exchanging routing information between autonomous systems. BGP is used to exchange routing information between multiple transit autonomous systems as well as between transit and stub autonomous systems. BGP uses path attributes to provide more information about each route and in particular to maintain an autonomous system (AS) path. An AS path includes the AS number of each autonomous system that the route has transited, which provides information sufficient to prevent routing loops in an arbitrary topology. Path attributes may also be used to distinguish between groups of routes to determine administrative preferences, allowing greater flexibility in determining route preference to achieve a variety of administrative ends. GateD supports version 4 of the BGP protocol.

BGP supports two basic types of sessions between neighbors: internal (sometimes referred to as IBGP) and external (EBGP). Internal sessions are run between routers in the same autonomous system. External sessions run between routers in different autonomous systems. When an AS sends routes to an external peer, the local AS number is prepended to the AS path. This means that routes received from an external peer are guaranteed to have the AS number of that peer at the start of the path. In general, routes received from an internal neighbor will not have the local AS number prepended to the AS path. Those routes will have the same AS path that the route had when the originating internal neighbor received the route from an external peer. Routes with no AS numbers in the path may be legitimately received from internal neighbors. These routes should be considered internal to the receiver's own AS.

External BGP sessions may or may not include a single metric, which BGP calls the Multi-Exit Discriminator (MED) among the path attributes. MED is a 32-bit unsigned integer. Smaller values of the MED are preferred. This metric is used only to break ties between routes with equal preference from the same neighboring AS.

Internal BGP sessions carry at least one metric in the path attributes, which BGP calls the **`Local_Pref`**. A route is preferred if its value for this metric is larger. Internal sessions may optionally include a second metric, the MED, carried in from external sessions. The use of these metrics is dependent on the type of internal protocol processing that is specified.

BGP collapses as many routes with similar path attributes as it can into a single update for advertisement. It also sends another update once the maximum packet size is reached. The churn caused by the loss of a neighbor will be minimized, and the initial advertisement sent during peer establishment will be maximally compressed. BGP does not read information from the kernel message by message, but fills the input buffer. BGP processes all complete messages in the buffer before reading again. BGP also does multiple reads to clear all

incoming data queued on the socket. This feature may cause other protocols to be blocked for prolonged intervals by a busy peer connection. All unreachable messages are collected into a single message and sent prior to reachable routes during a flash update. Another update is sent once the maximum packet size is reached.

Two internal routing groups exist: **group type internal** and **group type routing**. The **group type internal** expects all peers to be directly attached to a shared subnet so that, like external peers, the next hops received in BGP advertisements may be used directly for forwarding. But **group type routing** will determine the immediate next hops for routes by using the next hop received with a route from a peer, and using this next hop to look up an immediate next hop in an IGP's routes. Such groups support distant peers but need to be informed of the IGP whose routes they are using to determine immediate next hops.

For **group type internal** BGP, where possible, a single outgoing message is built for all group peers based on the common policy. A copy of the message is sent to every peer in the group. The copy includes possible adjustments to the next-hop field as appropriate to each peer. Another update is sent once the maximum packet size is reached. This process minimizes the computational load of running large numbers of peers in these types of groups.

BGP allows unconfigured peers to connect if an appropriate group has been configured with an **allow** clause.

14.2 BGP Syntax

```
bgp ( on | off )
    {
    [ clusterid host-id ; ]
    [ defaultmetric metric ; ]
    [ discard-nonprefixed-confederations |
      ignore-nonprefixed-confederations ]
    [ open-on-accept ; ]
    [ preference bgppreference ; ]
    [ traceoptions trace_options ; ]
    group type
        ( external peeras autonomoussystem
        | internal peeras autonomoussystem
        | routing peeras autonomoussystem proto protocol )
            [ ascount count ]              # external only
            [ comm community_values ]
            [ confed ]
            [ gateway host ]
            [ holdtime time ]
            [ ignorefirstashop ]           # external only
            [ keep ( all | none ) ]
```

```
        [ keepalivesalways ]
        [ localtcp local_address ]
        [ localas autonomous_system ]   # external only
        [ logupdown ]
        [ med ]
        [ metricout metric ]
        [ nexthopself ]                 # external only
        [ no-mp-nexthop ]
        [ noaggregatorid ]
        [ nogendefault ]
        [ nov4asloop ]
        [ outdelay time ]               # external only
        [ passive ]
        [ preference grouppreference ]
        [ preference2 grouppreference2 ]
        [ recvbuffer buffer_size ]
        [ reflector-client [ no-client-reflect ] ]
        # internal and routing types only
        [ routetopeer ]
        [ sendbuffer buffer_size ]
        [ setpref metric ]      # internal and routing types only
        [ showwarnings ]
        [ traceoptions trace_options ]
        [ ttl ttl ]                     # routing only
{
        #
        # There can be zero or one
        # "allow" clauses within a peer group.
        #
        allow {
            all ;
            | host ipnumber ;
            | classful network ;
            | network mask mask ;
            | network masklen number ;
            | network '/' number ;
        } ;
        #
```

```
                # There can be zero or more
                # "peer" clauses within a peer group.
                #
                peer host
                    [ ascount count ]
                    [ gateway host ]
                    [ holdtime time ]
                    [ ignorefirstashop ]
                    [ keep ( all | none ) ]
                    [ keepalivesalways ]
                    [ localtcp local_address ]
                    [ logupdown ]
                    [ med ]
                    [ metricout metric ]
                    [ nexthopself ]
                    [ no-mp-nexthop ]
                    [ noaggregatorid ]
                    [ nogendefault ]
                    [ nov4asloop ]
                    [ outdelay time ]
                    [ passive ]
                    [ preference peerpreference ]
                    [ preference2 peerpreference2 ]
                    [ recvbuffer buffer_size ]
                    [ reflector-client [ no-client-reflect ] ]
                    [ routetopeer ]
                    [ sendbuffer buffer_size ]
                    [ showwarnings ]
                    [ traceoptions trace_options ]
                    [ ttl ttl ]
                ;
                #
                # There should be at least one "allow" or "peer" clause
                # within a "group type" statement.
                #
        } ;
    } ;
```

Notes:

1. You must specify the Autonomous System and Router ID at the top of your configuration file in order for BGP to work.
2. One or more **group type** clauses must appear within the bgp statement. The **group type** clause is used to create peer groups that share common attributes.
3. Within the group type statement, the **allow** and/or **peer** clauses are used to specify the peers that are in the group.
4. The **allow** clause allows peering sessions to be established by hosts within one or more networks. The **allow** clause should appear at most only once within a peering group.
5. The **peer** clause is used to individually specify peers and to permit overriding specific peering group options.
6. The **peer** clause can appear multiple times within a peering group.

More detailed descriptions of these commands can be found on page 225 of the *Command Reference Guide*. See "Examples of Importation into Multicast RIBs" on page 142 for more information about importing and BGP. See "Exporting to BGP" on page 149 for more information about exporting and BGP.

14.3 Extended BGP-4 Features

The following features are provided in extended BGP-4:

- AS Path prepend - GateD allows the prepending of Autonomous Systems.
- Route reflection (RFC 2796) - Route Reflection is supported for reduction of large internal peer groups. See "Route Reflection Overview and Examples" on page 71.
- BGP Route Flap Damping (RFC 2439) - GateD supports the varied parameters on Route Flap Damping.
- Community Support (RFC 1997) - GateD allows for filtering of routes based on communities on import. In exporting routes, GateD allows the communities to be added or deleted. See "Communities Overview and Examples" on page 77.
- BGP Confederations (RFC 3065) - BGP Confederations allows multiple internal ASs to be used for scaling large networks. This confederation is represented as a single external AS.

14.4 Route Selection

BGP selects the best path to an AS from all the known paths and propagates the selected path to its neighbors.

14.5 Cisco® Interoperability

GateD configuration differs greatly from Cisco® routers. This section compares the following:

- BGP route selection
- Local_Pref configuration
- MED configuration
- import and export policy configuration

14.5.1 Cisco® vs. GateD Route Selection

The following table compares Cisco® 11.0/12.0 and GateD bgp-4-16 draft route selection policy:

Cisco® (11.0/12.0)	GateD (bgp-4-16 draft)
Active Route - If the next hop is inaccessible, do not consider it.	Active Route - If GateD cannot install a route in the kernel, GateD will not consider it (select the route as the active route).
Configured Policy - Consider larger BGP administrative weights first.	Configured Policy - Consider the route with smallest preference, as determined by the policy defined in `gated.conf`. Ties are broken by the **`preference2`** with the lowest values.
Local_Pref - If the routes have the same weight, consider the route with higher local preference.	Local_Pref - If the BGP Preferences match (**`preference`** and **`preference2`**), prefer the route with the highest BGP local preference.
Local Router - If the routes have the same local preference, prefer the route that the local router originated.	
Shortest AS Path - If no route is originated, prefer the shorter AS path.	Shortest AS Path - If the routes have the same BGP local preference, prefer the route with the fewest Autonomous Systems listed in its AS path.
IGP < EGP < Incomplete - If all routes have paths with the same AS path length, prefer the lowest origin code (IGP < EGP < Incomplete).	Origin IGP < EGP < Incomplete - If routes have the same AS path length, prefer the lowest origin code. Next in preference is the route with AS path origin of EGP. Least preferred is an AS path that is incomplete.

Cisco® (11.0/12.0)	GateD (bgp-4-16 draft)
MED - If origin codes are the same and all the paths are from the same Autonomous System, prefer the path with the lowest Multi Exit Discriminator (MED) metric. A missing metric is treated as zero.	MED (if not ignored) - If origin codes are the same, prefer the highest (worst) Multi-Exit Discriminator. MEDs are only compared between routes that were received from the same neighbor AS. This test is applied only if the local AS has two or more connections to a given neighbor AS. A missing metric is treated as the best (lowest) MED. MED comparison must be enabled; it is disabled by default.
External/Internal - If the MEDs are the same, prefer external paths over internal paths.	Source IGP < EBGP < IBGP - If the MEDs are the same, prefer first the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.
Closest Neighbor - If IGP synchronization is disabled and only internal paths remain, prefer the path through the closest neighbor.	Shortest IGP distance - If the IGP distances are the same, prefer the route whose next hop IP address is closer (with respect to the IGP distance)
Lowest IP Address - If the neighbors are equally close, prefer the route with the lowest IP address value for the BGP router ID.	Lowest Router ID - If the sources are the same, prefer the route whose next hop IP address is numerically lowest.

14.6 Local_Pref Configuration Example

The following configurations set a **`Local_Pref`** of 120 for peers in AS 200. Note that GateD's configuration uses the **`setpref`** command. The **`Local_Pref`** value comes from this equation: **`Local_Pref = 254 - (global protocol preference for this route) + metric`**. The global protocol preference for BGP is 170. (See "Assigning Preferences" on page 11.) In this example, we use the syntax **`setpref 36`** to specify a **`Local_Pref`** value of **`120`** (**`254-170+36 = 120`**) for BGP routes.

Cisco®:

```
router bgp 100
  network 192.168.0.0
  neighbor 192.168.1.1 remote-as 200
  neighbor 192.168.1.1 route-map set-local-pref in
route-map set-local-pref permit 10
  set local preference 120
```

GateD:

```
group type internal peeras 200 setpref 36 {  # (254-170+36) = 120
    peer 192.168.1.1;
};
```

14.6.1 MED Configuration Example

The following configurations set a metric of 127 on routes exported to AS 200.

Cisco®:

```
ip as-path access-list 1 permit .*
route-map med permit 10
match as-path 1
set metric 127
```

GateD:

```
export proto bgp as 200 {
    proto bgp aspath .* origin any {
        all metric 127;
    };
};
```

14.6.2 Import Filter Example

Cisco®:

```
router bgp 200
  neighbor 192.168.10.32 remote-as 100
  neighbor 192.168.10.32 filter-list 2 in
ip as-path access-list 2 deny _690$
ip as-path access-list 2 permit .*
```

GateD:

```
autonomoussystem 200;
routerid 192.168.10.55;
bgp on {
    group type external peeras 100 {
        peer 192.168.10.32;
    };
};
import proto bgp aspath (.* 690) origin any {
    all restrict;
};
import proto bgp aspath (.*) origin any {
    all;
};
```

14.6.3 Export Filter Example

Cisco®:

```
router bgp 200
 neighbor 192.168.10.32 remote-as 100
 neighbor 192.168.10.32 filter-list 3 out
ip as-path access-list 3 deny _400$
ip as-path access-list 3 permit .*
```

GateD:

```
autonomoussystem 200;
routerid 192.168.10.55;
bgp on {
    group type external peeras 100 {
        peer 192.168.10.32;
    };
};
export proto bgp as 100 {
    proto bgp aspath (.* 400) origin any {
        all restrict;
    };
    proto bgp aspath (.*) origin any {
        all;
    };
```

```
};
```

14.7 BGP Issues

14.7.1 Third-Party Route Advertisement

Third-party route advertisements are a special form of advertisements to peers. In particular, any advertisement that has a next hop attribute that contains an IP address different from the IP address of the peer sending the advertisement is called "third party." A third-party advertisement is legal, essentially, when the next hop that is advertised is on the network that is used for peering.

GateD, by default, performs third party route advertisements when the next hop that it is using, if used, would result in a legal third-party route advertisement. Also, by default, GateD rejects third-party route advertisements that are illegal. There are two options that can be used to alter this default behavior: **nexthopself** and **gateway**. The former deals with routes originated by GateD, the latter with both sending and receiving third-party advertisements.

nexthopself causes GateD to include a next hop of its own IP address in all advertisements to an external peer. Hence, no advertisements that GateD sends could be considered third party.

The second option, **gateway**, is meant for use in situations where the peers are not directly connected to one another. With the **gateway** option, you specify the first hop along the path to the peer. GateD will then perform third-party route advertisements as though the network shared with the gateway were really the network shared with the peer. GateD will also substitute, on received advertisements, the address of the gateway for the address of the next hop received.

The following is a sample BGP statement in which GateD turns off third-party route advertisements with respect to peer 192.168.10.1, but not with respect to 192.168.10.2.

```
bgp yes {
    group type external peeras 1 {
        peer 192.168.10.1 nexthopself;
        peer 192.168.10.2;
    };
};
```

In the preceding example, if GateD learned reachability for network 192.168.20 with a next hop of 192.168.10.100, the advertisements to peer 192.168.10.1 and peer 192.168.10.2 would differ: the advertisement to peer 192.168.10.1 would contain a next hop of the GateD box, and the advertisement to peer 192.168.10.2 would contain a next hop of 192.168.10.100.

And here is an example where the GateD box is attached to the network 192.168.10/24, but the peer is not. Note that the gateway router (192.168.10.1) must be able to forward packets to the peer (192.168.77.12).

```
bgp yes {
    group type external peeras 1 {
        peer 192.168.77.12 gateway 192.168.10.1;
```

```
        };
    };
```

In this example, GateD will ensure that all of the next hops that it advertises to its peer (192.168.77.12) are on the network shared with the gateway (192.168.10/24). Any next hops that it receives from the peer (192.168.77.12) will be replaced with the address of the gateway (192.168.10.1).

14.7.2 Determining Next Hops

In GateD, at present, there are three different cases for next hop determination: **`group type internal`**, **`group type external`**, and anything else. Modification of the next hop for **`group type external`** is covered in "Third-Party Route Advertisement" on page 70. As far as IBGP peers are concerned, the BGP specification is clear: the next hop that is sent shall be the next hop that was received.

`group type internal` is intended for peers on directly attached networks. If the peers are not on directly shared networks, **`group type routing`** should be used.

For next hop determination, **`group type routing`** uses essentially the same algorithm that external peers with the **`gateway`** option use. GateD determines which network is being used to reach the immediate next hop to its peer. It then ensures that the next hop advertised is on the same network as the immediate next hop.

14.7.3 AS Path Stuffing and Spoofing

AS Path "stuffing" or "prepending" is accomplished with the **`ascount`** command. **`ascount`** is used to bias upstream peers' route selection. (Most routers prefer routes with shorter AS Paths.)

In previous versions of BGP, the specification had not allowed the existence of looped AS Paths. Loops must be ignored in order to allow AS prepending. The **`localas`** command can be used to spoof the AS that BGP represents to a group of peers. The default AS is that configured in the **`autonomoussystem`** statement. **`localas`** provides a way to speak BGP from more than one AS.

14.7.4 Route Reflection Overview and Examples

Generally, all border routers in a single AS need to be internal peers of each other, and, in fact, all non-border routers frequently need to be internal peers of all border routers. Although this configuration is usually acceptable in small networks, it may lead to unacceptably large internal peer groups in large networks. To help address this problem, BGP supports route reflection for internal peer groups. When using route reflection, the rule that a router may not readvertise routes from internal peers to other internal peers is relaxed for some routers, called "route reflectors". A typical use of route reflection might involve a "core" backbone of fully meshed routers (all the routers in the group peered directly with all other routers in the group). Some of these routers act as route reflectors for routers that are not part of the core group.

Two types of route reflection are supported: routes can be sent to all internal peers or only to internal peers that are not members of the client's group. By default, all routes received by the route reflector from a client are sent to all internal peers (including the client's

group, but not the client itself). If the **no-client-reflect** option is enabled, routes received from a route reflection client are sent only to internal peers that are not members of the client's group. In this case, the client's group must itself be fully meshed. In either case, all routes received from a non-client internal peer are sent to all route reflection clients.

Typically, a single router will act as the reflector for a set (or cluster) of clients. However, for redundancy, two or more can also be configured to be reflectors for the same cluster. In this case, a cluster ID should be selected using the **clusterid** keyword to identify all reflectors serving the cluster. Gratuitous use of multiple redundant reflectors is not advised, because it can lead to an increase in the memory required to store routes on the redundant reflectors' peers.

No special configuration is required on the route reflection clients. From a client's perspective, a route reflector is simply a normal IBGP peer. Any BGP version 4 speaker should be able to be a reflector client.

Refer to the route reflection specification document (RFC 1966) for further details. RFC 1966 can be found at:

http://www.ietf.org/rfc/rfc1966.txt

All routes received from any group member will be sent to all other internal neighbors, and all routes received from any other internal neighbors will be sent to the reflector clients. Because the route reflector forwards routes in this way, the reflector-client group need not be fully meshed. If the **no-client-reflect** option is specified, routes received from reflector clients will only be sent to internal neighbors that are not in the same group as the sending reflector client. In this case, the reflector-client group should be fully meshed. In all cases, routes received from normal internal peers will be sent to all reflector clients.

Note: It is necessary to export routes from the local AS back into the local AS when acting as a route reflector. For example, suppose that the local AS number is 2. An export statement like the following would suffice to make reflection work correctly.

```
export proto bgp as 2 {
    proto bgp as 2 {all;}; # for reflection
    # other exports
};
```

If the cluster ID is changed and GateD is reconfigured with a **SIGHUP**, all BGP sessions with reflector clients will be dropped and restarted.

Another example follows.

```
traceoptions "/var/tmp/gated.log" replace size 1000k files 3 all;
autonomous-system 64512;
routerid 192.168.11.1;
bgp yes {
    group type internal peeras 64512 reflector-client {
        peer 192.168.10.2;
        peer 192.168.10.3;
```

```
        peer 192.168.10.4;
        peer 192.168.10.5;
        peer 192.168.10.6;
    };
    group type internal peeras 64512 {
        peer 192.168.11.2;
        peer 192.168.11.3;
    };
};

static {
    default gw 172.16.0.1 retain;
};

import proto bgp as 64512 {
    all;
};

export proto bgp as 64512 {
    proto bgp as 64512 {
        all;
    };
};
```

14.7.5 Weighted Route Damping Overview, Syntax, and Defaults

The basic idea of weighted route damping is to treat routes that are being announced and withdrawn (flapping) at a rapid rate as unreachable.

If a route flaps at a low rate, it should not be suppressed at all, or suppressed only for a brief period of time. With weighted route damping, the suppression of a route or routes occurs in a manner that adapts to the frequency and duration that a particular route appears to be flapping. The more a route flaps during a period of time, the longer it will be suppressed. The adaptive characteristics of weighted route damping are controlled by a few configurable parameters.

Currently, only routes learned via BGP are subject to weighted route damping, although no protocols will announce suppressed routes. The weighted route damping configuration statement is not within the BGP statement but is a separate and distinct configuration; conceptually, it is much like **`interface`** or **`kernel`** statements. (Refer to “Chapter 7 Interface Statement” on page 23 and “Chapter 18 Kernel Interface” on page 95 for more information.)

The syntax for weighted route damping in GateD is:

```
dampen-flap {
```

```
            [ suppress-above metric ; ]
            [ reuse-below metric ; ]
            [ max-flap metric ; ]
            [ unreach-decay time ; ]
            [ reach-decay time ; ]
            [ keep-history time ; ]
        };
```

suppress-above *metric*

suppress-above is the value of the instability metric at which route suppression will take place (a route will not be installed in the FIB or announced even if it is reachable during the period that it is suppressed).

reuse-below *metric*

reuse-below is the value of the instability metric at which a suppressed route will become unsuppressed, if it is reachable but currently suppressed. The value assigned to **reuse-below** must be less than **suppress-above**.

max-flap *metric*

max-flap is the upper limit of the instability metric. This value must be greater than the larger of 1 and **suppress_above**.

Assigned to the above three parameters is a floating point number in units of flaps. Each time a route becomes unreachable, 1 is added to the current instability metric.

reach-decay *time*

reach-decay specifies the time desired for the instability metric value to reach one half of its current value when the route is reachable. This half-life value determines the rate at which the metric value is decayed. A smaller half-life value will make a suppressed route reusable sooner than a larger value.

unreach-decay *time*

unreach-decay acts the same as **reach-decay** except that it specifies the rate at which the instability metric is decayed when a route is unreachable. It should have a value greater than or equal to **reach-decay**.

keep-history *time*

keep-history specifies the period over which the route flapping history is to be maintained for a given route. The size of the configuration arrays described below is directly affected by this value.

If only **dampen-flap {};** is specified in the configuration file, then the following default values are used:

```
suppress-above = 3.0;
reuse-below = 2.0;
max-flap = 16.0;
unreach-decay = 900;
reach-decay = 300;
```

```
keep-history = 1800
```

14.7.6 Setpref/Local_Pref Overview

Note: The term "preference" as used in **setpref/Local_Pref** is not the same as each protocol's **preference** in GateD. Each protocol has a parameter, **preference**, that specifies how active routes will be selected. When a route has been learned from more than one protocol, the active route will be selected from the protocol with the lowest **preference**. Each protocol has a default preference in this selection. **setpref/Local_Pref** is BGP-specific and does not influence how active routes from BGP will compare to those learned from other protocols.

The **setpref** option allows GateD to set the **Local_Pref** to reflect GateD's own internal preference for the route, as given by the global protocol preference value (which can be found at "Preference Selection Precedence" on page 12). **Local_Pref** can be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. The **setpref** option can be used with routing or internal type groups. The **Local_Pref** is never set directly, but rather as a function of the GateD **preference** and **setpref** metrics.

If the **setpref** option is set on one internal peer group, it must be set on all internal peer groups. The **setpref** option may be used only on internal peer group types (internal or routing).

The translation of GateD's internal preference to and from **Local_Pref** is done as follows. In the table below, *metric* is the argument to **setpref**. (For example, in the statement, "**setpref** *100*," *metric* is 100.) "Exported Preference" is the GateD preference of the exported route. "Imported Preference" is the GateD preference assigned to the imported route.

Exported Preference	Local_Pref	Imported Preference
Less than *metric*	254	*metric*
metric to 254	254 to *metric*	*metric* to 254
N/A	Greater than 254	*metric*

In effect, any GateD preference of less than *metric* is exported such that it will be re-imported (by a distant GateD) with a preference of exactly *metric*. Any preference of *metric* or above will be exported such that it will be re-imported with the same preference it had originally.

Local_Pref, as exported to BGP peers, is calculated as:

Local_Pref = 254 - (global protocol **preference** for this route) + *metric*

A value greater than 254 will be reset to 254. GateD will only send **Local_Pref** values between 0 and 254. For example, suppose GateD is sending routes to an internal group using "**setpref** *100*," and the routes are subsequently received by another router in the group, also using "**setpref** *100*."

The table below lists some sample route preferences, the **Local_Prefs** with which the routes will be sent, and the preferences with which the routes will be imported.

Preference Before Export	Local_Pref	Preference After Import
170	184=(254-170+100)	170
171	183	171
254	100	254
100	254	100
5	254	100

Notes:

- Non-GateD IBGP implementations may send **Local_Prefs** that are greater than 254. When operating a mixed network of this type, it is recommended that all routers restrict themselves to sending **Local_Prefs** in the range *metric* to 254.
- All routers in the same network that are running GateD and participating in IBGP should use **setpref** uniformly. That is, if one router has **setpref** set, all should set it, and all should use the same value of *metric*. The value for *metric* should be selected to be consistent with the import policy in use in the network. For example, if import policy sets GateD preferences ranging from 170 to 200, a **setpref** *metric* of 170 would make sense. It is advisable to set *metric* high enough to avoid conflicts between BGP routes and IGP or static routes.

Routes propagated by IBGP must include a **Local_Pref** attribute. **Local_Pref** may be used by a BGP speaker to inform other BGP speakers in its own autonomous system of the originating speaker's degree of preference for an advertised route. Unless the **setpref** option has been set, BGP sends the **Local_Pref** path attribute as 100.

GateD always uses the received **Local_Pref** to select between BGP routes that have the same GateD preference. BGP routes with a larger **Local_Pref** are preferred.

For this topology:

The following configuration will cause AS 65100 to prefer routes from the BGP1--BGP2 link.

BGP1 Configuration

```
bgp yes {
    group type external peeras 65000 {
        peer 10.0.0.2; # BGP2
    };
    group type internal peeras 65100 setpref 100 {
        peer 192.168.10.2; # BGP3
    };
};
```

BGP3 Configuration

```
bgp yes {
    group type external peeras 65000 {
        peer 10.0.0.2; # BGP2
    };
    group type internal peeras 65100 setpref 99 {
        peer 192.168.10.1; # BGP1
    };
};
```

14.7.7 Communities Overview and Examples

The community attribute allows the administrator of a routing domain to tag groups of routes with a community tag. Using communities allows the administrator to limit the routes that can be imported or exported. The tag consists of 2 octets of AS and 2 octets of community ID. The **community** attribute is passed from routing domain to routing domain to maintain the grouping of these routes. A set of routes may have more than one community tag in its **community** attribute.

The import and export policy of a community is configured using the **comm** clause (or **comm-add** clause) on the **group**, **import**, and **export** statements.

Please refer to the communities specification (RFC 1997) and its accompanying usage document (RFC 1998) for further details on BGP communities. RFC 1997 can be found at:

http://www.ietf.org/rfc/rfc1997.txt

RFC 1998 can be found at:

http://www.ietf.org/rfc/rfc1998.txt

Communities can be specified as an AS and a community ID (with the **comm-split** keyword) or as one of the distinguished special communities (with the **comm** keyword). When originating BGP communities, the set of communities that is actually sent is the union of the communities received with the route (if any), those specified in group policy (if any), and those specified in export policy (if any). When receiving BGP communities, the update is matched only if all communities specified in **comm** are present in the BGP update. (If additional communities are also present in the update, it will still be matched.) The limit of 25 communities in any single policy clause can be increased at compile time by increasing the value of **AS_COMM_MAX**.

comm-split *autonomous_system community_id*

comm-split causes a community "tag" to be added to the transmitted path attributes. The *autonomous_system* part of the community should be set to the local AS, unless there is a specific need to do otherwise. This associates an AS with a community.

community no-export

community no-export is a special community that indicates that the routes associated with this attribute must not be advertised outside a BGP AS boundary.

community no-advertise

community no-advertise is a special community that indicates that the routes associated with this attribute must not be advertised to other BGP peers.

community no-export-subconfed

community no-export-subconfed is a special community that indicates that the routes associated with this attribute must not be advertised to external BGP peers.

community none

community none is not actually a community, but rather a keyword that specifies that a received BGP update is only to be matched if no communities are present. It has no effect when originating communities.

The following example will import only routes from AS 203 that are stamped with community 99:

```
import proto bgp as 203
    comm {
        comm-split 203 99
    }
{
```

```
        all;
    };
```

The following example will export only routes to AS 205 and from AS 203 that are stamped with community 99:

```
    export proto bgp as 205
        comm {
            comm-split 203 99
        }
    {
        proto bgp static {
            all;
        };
    };
```

Communities are added to a route on export with the **comm-add** aspath options.

```
    export proto bgp as 205
        comm-add {
            comm-split 203 99
        }
    {
        proto bgp static {
            all;
        };
    };
```

14.7.8 Multi-Exit Discriminator Overview and Examples

The Multi-Exit Discriminator (MED) allows the administrator of a routing domain to choose between various exits from a neighboring AS. This attribute is used only for decision-making in choosing the best route to the neighboring AS. If all the other factors for a path to a given AS are equal, the path with the lower MED value takes preference over other paths.

This attribute, if learned from an external AS, can be propagated only to internal peers, unless you are in a BGP Confederation. The MED value can be propagated to BGP Confederation external peers. The MED value is propagated only if the **med** keyword is specified on the BGP peers or group.

The MED attribute for BGP version 4 is a four-byte unsigned integer. MED is originated using the **metricout** option of group or peer statements or the **metric** option of the export statement. It is imported using the **med** keyword on the BGP group statement.

The **metricout** and **metric** options are used to specify the value of MED for exported routes. The **metricout** option can be specified on the group statement:

```
group type external peeras 31337 metricout 5 {
    peer 192.168.10.32;
    peer 192.168.10.33;
};
```

It can also be specified on the peer statement:

```
group type external peeras 31337 {
    peer 192.168.10.32 metricout 2;
    peer 192.168.10.33 metricout 3;
};
```

The equivalent metric keyword can be specified on the export statement like this:

```
export proto bgp as 31337 metric 5 {
    proto static {
        all;
    };
};
```

And like this:

```
export proto bgp as 31337 {
    proto bgp as 64000 metric 1 {
        all;
    };
    proto static metric 3 {
        all;
    };
    proto direct metric 7 {
        all;
    };
};
```

The **med** keyword must be specified on the group statement for GateD to consider metrics when calculating a next hop (the default action is to ignore MEDs).

14.7.9 Confederations

The BGP specification requires that all internal BGP speakers maintain a full mesh. As the number of BGP speakers in an AS grows, the number of peering sessions that must be maintained grows factorially. This can put a great strain on infrastructure both in terms of the hardware in routers and in terms of the amount of bandwidth consumed by routing traffic.

In order to help relieve the strain on resources, RFC 3065 specifies an alternative to full mesh IBGP known as "BGP Confederations." A BGP Confederation is a collection of autono-

mous systems that present themselves as a single AS to peers outside of the confederation. RFC 3065 can be found at:

http://www.ietf.org/rfc/rfc3065.txt

All BGP speakers within a confederation are assigned two AS numbers. The first of these is their normal AS number to be used within the confederation. The second is known as their confederation ID.

All BGP speakers within a single confederation must be assigned the same confederation ID. This confederation ID is the autonomous system number that BGP speakers outside of the confederation see as consisting of all BGP speakers within the confederation, despite the fact that the various members of the confederation can be within different autonomous systems.

When BGP speakers within the same confederation communicate with each other, they perform identically to BGP speakers not in confederations with one exception: rather than using the **AS_SEQUENCE** and **AS_SET** path attributes, they use the **CONFED_SEQUENCE** and **CONFED_SET** path attributes. Then, when a BGP speaker within the confederation goes to advertise routes to a BGP speaker not within the confederation, all attributes of the type **CONFED_SEQUENCE** or **CONFED_SET** are stripped and replaced with a single **AS_SEQUENCE** consisting of the confederation identifier. In this fashion, the internal AS topology of the confederation is kept hidden from the rest of the world.

The following **gated.conf** shows a confederation border router. It has two peers outside of the confederation, one inside the confederation, and some confederation internal peers.

```
autonomoussystem 64512;
confed-id 100;
bgp yes {
    group type routing peeras 64512 confed proto ospf {
        peer 192.168.1.1 ;
        peer 192.168.1.4 ;
    } ;
    group type external peeras 65000 confed {
        peer 10.132.10.1 ;
    } ;
    group type external peeras 200 {
        peer 172.16.50.1 ;
    } ;
} ;

# Import everything from our internal confederation peers
    import proto bgp as 64512 {
        all ;
    } ;
```

```
# Import everything from our external confederation peer
    import proto bgp as 65000 {
        all ;
    } ;

# Import everything from our external non-confederation peer
    import proto bgp as 200 {
        all ;
    } ;

# Redistribute everything from our external non-confederation and our
# external confederation peer to our internal peers. Note that we are
# not operating as a route reflector, so we do not redistribute routes
# from our internal peers to our other internal peers.

    export proto bgp as 64512 {
        proto bgp as 200 {
            all ;
        } ;
        proto bgp as 65000 {
            all ;
        } ;
    } ;

# Redistribute our routes from our external confederation peer to our
# internal confederation peers and our external non-confederation
# peer.
    export proto bgp as 65000 {
        proto bgp as 200 {
            all ;
        } ;
        proto bgp as 64512 {
            all ;
        } ;
    } ;

# We want to receive traffic for this AS on our external links, so
# propogate everything from our confederation.

    export bgp as 200 {
```

```
        proto bgp as 64512 {
            all ;
        } ;
        proto bgp as 65000 {
            all ;
        } ;
    } ;
```