Routing TCP/IP Vol 2 Notes

13 Jul 2008

Chapter 2: Introduction to BGP4

Border Gateway Protocol (BGP) was developed to replace *Exterior Gateway Protocol (EGP)*. The current incarnation of the protocol is version four, introduced in **RFC 1771**.

BGP is typically only required for Internet connectivity, and even then often only when peering with multiple service providers.

BGP behaves as a distance vector protocol, but uses an AS path instead of a single metric. For this reason, it is referred to as *path vector*.

Load balancing (by default, across a maximum of 6 paths) is only provided by eBGP; iBGP can only use one link.

BGP message types:

Open - Used to form peer relationships

Keepalive - Periodic maintenance of relationships

Update - Communicates routing information

Notification - Communicates an error

BGP neighbor states:

Idle

Connect - A TCP connection is being attempted

Active - A TCP connection has failed; the router is waiting to be contacted by its peer

OpenSent - TCP session established, open message sent

OpenConfirm - Waiting for a keepalive from the peer

Established

Path Attributes

Attribute classes:

Well-known mandatory attributes must be supported and included

Well-known discretionary attributes must be supported but may not be included

Optional transitive attributes don't have to be supported, but must be passed onto peers

Optional nontransitive attributes don't have to be supported, and can be ignored

Attributes:

Origin (WM) - The source of the route (IGP > EGP > unknown)

AS Path (WM) - An ordered list of the ASs the route has traversed

Next Hop (WM) - Specifies the next-hop address for the route

Local Preference (WD) - Communicated between iBGP peers to favor a route out of the AS

Multi Exit Discriminator (ON) - Advertised to eBGP peers to indicate a preferred entrance into the local AS

Atomic Aggregate (WD) - Notes that route summarization has been performed

Aggregator (OT) - Identifies the router and AS where summarization was performed

Community (OT) - Provides route tagging capability

Originator ID (ON) - Identifies a route reflector

Cluster List (ON) - Records the route reflector clusters the route has traversed

Administrative *weight* is a Cisco proprietary attribute, a 16-bit value referenced only by the local router.

An AS Path can be one of two types (as distinguished by its type code):

AS Sequence - An ordered list

AS Set - An unordered list

An AS Set is used to record AS numbers lost when aggregation is performed. The Atomic Aggregate attribute does not have to be included to indicate aggregation has been performed if an AS Set is included.

BGP Operation

Decision Process

1. Prefer the route with the highest administrative weight
2. If weights are equal, select highest local preference
3. If local preferences are equal, prefer locally originated routes
4. If origins are equal, prefer the shortest AS Path
5. If AS Path lengths are equal, prefer the most favorable origin code (IGP > EGP > incomplete)
6. If origin codes are equal, prefer lowest MED (only if all candidates routes are advertised from the same AS)

7. If MEDs are equal, prefer eBGP > eBGP confederation > iBGP
8. If the route types are equal, prefer the route with the lowest IGP metric to its next hop
9. If IGP metrics are equal and are from the same AS, load balance
0. If multipath is not enabled, select the route with the lowest BGP router ID

Route Dampening

Route dampening reduces the effects of flapping routes by preventing their propagation through a network.

A route is assigned a *penalty* when it flaps. This penalty increases with the rate of flapping.

The penalty is decreased gradually. The time it takes to decrease by half is its *half-life*.

When the accumulated penalty exceeds the *suppress limit*, the route is suppressed. The route is put back in use when the penalty drops below the *reuse limit*.

The *maximum suppress limit* defines a maximum suppress time.

iBGP Synchronization

iBGP peers must be fully meshed, as iBGP-learned routes are not passed to other iBGP peers.

The rule of *synchronization* requires that an iBGP-learned route must be known by an IGP before it enters the BGP routing table.

The synchronization requirement can be disabled with `no synchronization`.

Managing Large-Scale Peering

Peer groups can be defined to simplify assigning characteristics to similar BGP neighbors.

In addition to simplifying configuration, peer groups improve performance by requiring fewer consultations of the policy database.

Communities can be implemented to apply policies to a group of routes (by appending one or more Community attributes).

Route reflectors can alleviate the iBGP relationships needed within an AS by purposefully relaying routes between iBGP peers. A route reflector and its clients is referred to as a *cluster*.

Route reflectors employ the Originator ID and Cluster List attributes to avoid loops within the AS.

A *confederation* is an AS which has been divided into sub-autonomous systems (*members*).

Like regular ASs, confederations use two types of AS Path for loop avoidance:

AS Confed Sequence - An ordered list of member ASNs

AS Confed Set - An unordered list of member ASNs

These confederation-specific attributes are not communicated outside the confederation.

Within a confederation, eBGP routes external to the confederation are preferred over eBGP routes from another confederation member AS.

Chapter 3: Configuring and Troubleshooting BGP

Basic configuration:

```
Router(config)# router bgp <ASN>
Router(config-router)# neighbor <address> remote-as <ASN>
```

The BGP router ID can be taken from one of the following sources, listed in order of preference:

1. The `bgp router-id` command
2. The highest IP address of a loopback interface
3. The highest IP address of a physical interface

BGP will summarize to classful boundaries by default. As in other protocols, this is disabled with `no auto-summary`.

Manually specifying routes to be advertised in BGP:

```
Router(config-router)# network <subnet> [mask <mask>]
```

Enabling redistribution into BGP:

```
Router(config-router)# redistribute <protocol ...>
```

Manually specified routes (by the `network` command) are marked as originating from an IGP ("i"), whereas redistributed routes are marked with an incomplete origin ("?").

A BGP router can be configured to advertise a default route to its neighbors:

```
Router(config-router)# neighbor <address> default-originate
```

The synchronization requirement can be disabled, allowing BGP to import internal routes not already known by an IGP into the routing table:

```
Router(config-router)# no synchronization
```

The next hop attribute for routes advertised to iBGP peers can be changed to the advertising router:

```
Router(config-router)# neighbor <address> next-hop-self
```

BGP timers can be adjusted (in seconds):

```
Router(config-router)# bgp timers <keepalive> <hold>
```

The source address from which a neighbor relationship is formed can be explicitly set (especially necessary for sourcing relationships from loopback interfaces):

```
Router(config-router)# neighbor <address> update-source <interface>
```

eBGP multihop must be enabled to increase the default TTL of eBGP packets from the default of 1:

```
Router(config-router)# neighbor <address> ebgp-multihop <TTL>
```

eBGP multihop is required if using loopback interfaces.

Route Aggregation

There are two methods to advertise aggregated routes: static routes and the `aggregate-address` command.

Using static routes:

```
Router(config)# ip route 192.168.0.0 255.255.0.0 Null0
...
Router(config-router)# network 192.168.0.0 mask 255.255.0.0
```

Using an aggregate address:

```
Router(config-router)# aggregate-address 192.168.0.0 255.255.0.0 [summary-only]
```

If the `summary-only` keyword is included, only the aggregate address is advertised. If it is omitted, the more-specific routes will be advertised as well as the aggregate.

Alternatively, a number of selected more-specific routes can be suppressed with a *suppress map*:

```
Router(config-router)# aggregate-address 192.168.0.0 255.255.0.0 suppress-map <route-map>
```

An aggregate address can also be assigned an *attribute map* to modify its attributes (for example, to set its origin):

```
Router(config-router)# aggregate-address 192.168.0.0 255.255.0.0 attribute-map <route-map>
```

By default, no AS Set is included with the aggregate route. To include an AS Set:

```
Router(config-router)# aggregate-address 192.168.0.0 255.255.0.0 as-set
```

When the AS Set is included, the aggregate route inherits all the attributes of its aggregated routes. An *advertise map* can be included so that the aggregate only inherits attributes from selected routes:

```
Router(config-router)# aggregate-address 192.168.0.0 255.255.0.0 as-set advertise-map
 <route-map>
```

Managing BGP Connections

A human-friendly description can be added to each BGP neighbor for clarity:

```
Router(config-router)# neighbor 192.168.123.45 description R7 in Atlanta
```

Password authentication can be enabled per neighbor (an MD5 hash is included in BGP packets):

```
Router(config-router)# neighbor 192.168.123.45 password FooBar
```

An *advertisement interval* can be configured per neighbor, to specify the time to wait between sending

advertisements (0 to 600 seconds):

```
Router(config-router)# neighbor 192.168.123.45 advertisement-interval <seconds>
```

BGP version negotiation can be disabled by statically declaring the version to be used:

```
Router(config-router)# neighbor 192.168.123.45 version <version>
```

A router can be configured to remove consideration of the AS Path length from the BGP decision process:

```
Router(config-router)# bgp bestpath as-path ignore
```

The number of prefixes received from a neighbor can be limited:

```
Router(config-router)# neighbor 192.168.123.45 maximum-prefix <count> [<warning threshold>]
 [warning-only]
```

The warning threshold percentage defines what percentage of the maximum must be reached before generating a warning message.

The `warning-only` keyword allows the connection to be maintained even if the peer exceeds the maximum.

The `neighbor shutdown` command can be used to temporarily disable a peer without deleting its configuration.

Routing Policies

Distribute lists can be applied to neighbors to restrict the routes advertised or accepted:

```
Router(config-router)# neighbor <neighbor> distribute-list <ACL> {in | out}
```

Filter lists can be applied to filter routes based on their AS Path:

```
Router(config)# ip as-path access-list 1 permit <regex>
...
Router(config-router)# neighbor <neighbor> filter-list <list> {in | out }
```

Route maps can be used to achieve the same functionality as distribute and filter lists, but provide greater flexibility as well as the option to modify attributes.

Applying a route-map to a neighbor:

```
Router(config-router)# neighbor <neighbor> route-map <route-map> {in | out}
```

An administrative weight (0-65535) can be locally assigned to routes from specific neighbors:

```
Router(config-router)# neighbor <neighbor> weight <weight>
```

Locally-originated routes receive a weight of 32,768 by default whereas all others have a weight of 0.

Weight can be assigned to only selected routes by implementing a route-map, or by appending the `weight` keyword after the `filter-list` parameter:

```
Router(config-router)# neighbor <neighbor> filter-list <list> weight <weight>
```

External BGP routes have an administrative distance of 20. Internal and local (originated by the local router with the `network` command) have a distance of 200.

A *backdoor link* is a private link between ASs which should be favored over eBGP routes. For this to work, the administrative distance of the eBGP route must be artificially inflated by inserting the distant network in the BGP process with the `backdoor` keyword:

```
Router(config-router)# network <network> backdoor
```

The BGP route to the specified network will be treated as local and will have an administrative distance of 200, allowing IGP-learned routes to be favored instead.

Local Preference (32-bit, default 100) is communicated among iBGP peers. It can be set with `ip default local-preference` or with `set local-preference` in a route-map.

The MED (or "metric") can be set to influence the path a neighboring AS takes into the local AS.

`set metric-type internal` can be used in a route-map to set the MED to the IGP's metric for the route.

`bgp always-compare-med` can be used under the BGP process to force a router to compared the MED of multiple routes to the same destination even if they originate from different ASs.

It is possible to influence the routing decisions of autonomous systems beyond directly connected neighbors by artificially lengthening the AS Path of routes. This is done by using `set as-path prepend` in a route-map.

Typically the local AS is prepended one or more times. For example, routers AS in 123 might use the following:

```
route-map PREPEND_AS permit 10
 match ip address <ACL>
 set as-path prepend 123 123 123
```

Route tagging is useful for preserving BGP information across redistribution into and out of an IGP.

By default, BGP routes redistributed into an IGP are tagged with their AS Path. This can be extended to include the origin by using `set automatic-tag` in a route map.

To automatically set the AS Path from the IGP tag when redistributing back into BGP, use `set as-path tag` in a route-map.

Route tags can also be used to preserve BGP communities.

To enable BGP route dampening:

```
Router(config-router)# bgp dampening [<half-life> <reuse> <suppress> <max-suppress>]
```

`show ip bgp dampened-paths` will display suppressed routes. `show ip bgp flap-statistics` displays all dampened routes as well as routes with a history of dampening.

`clear ip bgp dampening` will put suppressed routes back into service. `clear ip bgp flap-statistics` will additionally clear all flapping histories.

Large-Scale BGP

Peer group definition example:

```
Router(config-router)# neighbor BRANCHES peer-group
Router(config-router)# neighbor BRANCHES ebgp-multihop 2
Router(config-router)# neighbor BRANCHES update-source Loopback0
Router(config-router)# neighbor 10.1.0.84 peer-group BRANCHES
Router(config-router)# neighbor 10.1.0.84 remote-as 123
```

Policies applied to individual peer group members take precedence over those applied to the whole

group.

Communities

BGP communities can be set in a route-map with `set community <community>`. Additionally, each neighbor or peer group must be configured with `send-community` for the communities to be included in advertisements.

Well-known communities:

> **no-export** - Included with a route advertised to eBGP peers to instruct them not to advertise it beyond the neighbor AS
>
> **no-advertise** - Sent to iBGP peers to prevent them from propagating the route
>
> **local-as** - Used to restrict the propagation of a route to within a confederation

Custom communities can be set in decimal format or in AA:NN (ASN:number) format.

Defining a standard *community list*:

```
Router(config)# ip community-list 1 {permit | deny} <community> [<community> ...]
```

An extended community list:

```
Router(config)# ip community-list 101 {permit | deny} <regex>
```

Community lists can be matched by using `match community` in a route-map.

Communities can be added to a route without disturbing any communities already present by appending the `additive` keyword to the `set community` statement.

Specific communities matching a community list can be deleted with `set comm-list <number> delete`.

Private AS Numbers

The private ASNs are 64512 through 65535.

The `remove-private-as` parameter can be applied to a neighbor to strip all private ASNs from the AS Path of routes received from that neighbor:

```
Router(config-router)# neighbor <neighbor> remove-private-as
```

Confederations

A *confederation* is an AS which has been divided into smaller sub-ASs, which typically use private ASNs. eBGP is run between member ASs.

Routers within a confederation must be configured to recognize it as such:

```
Router(config-router)# bgp confederation identifier <ASN>
```

Additionally, routers which peer with other member ASs must be configured to recognize that the peer AS belongs to the same confederation:

```
Router(config-router)# bgp confederation peers <ASN> [<ASN> ...]
```

Route Reflectors

A route reflector must be configured to recognize which internal peers it should reflect routes to:

```
Router(config-router)# neighbor <neighbor> route-reflector-client
```

A route reflector will add an Originator ID, identifying the originator of the route, and a Cluster List, identifying the reflection cluster for loop avoidance.

By default, a route reflector enters its router ID in the Cluster List. A cluster ID can be manually specified with `bgp cluster-id`. This is necessary when there are multiple reflectors in a cluster.

If reflector clients are fully meshed, `no bgp client-to-client reflection` can be used to disable route reflection between clients. Routes from outside the cluster will still be reflected normally.

Chapter 4: Network Address Translation

NAT is described in **RFC 3022**.

There are four types of NAT address:

Inside Local - Inside addresses as seen from the inside

Inside Global - Inside addresses as seen from the outside

Outside Global - Outside addresses as seen from the outside

Outside Local - Outside addresses as seen from the inside

NAT entries can be static or dynamic.

The default translation timeout of 86,400 seconds (24 hours) can be changed with `ip nat translation timeout`.

Port Address Translation (PAT), or NAT overloading, maps multiple internal hosts to a single outside address using layer 4 port numbers.

NAT/PAT can also be used for load balancing and server virtualization.

NAT Issues:

- The TCP or UDP header checksum must be recalculated
- Fragments must be tracked
- NAT must be able to recognize protocols which embed addresses beyond the IP header
- NAT cannot alter IP addresses in encrypted data

Configuring NAT

Designating inside and outside interfaces:

```
Router(config-if)# ip nat {inside | outside}
```

Defining static assignments:

```
Router(config)# ip nat inside source static <inside local> <inside global>
Router(config)# ip nat outside source static <outside global> <outside local>
```

To allow static mappings of one inside local address to multiple inside global addresses, append the `extendable` keyword to the statement.

Creating an address pool for dynamic NAT:

```
Router(config)# ip nat pool <name> <start address> <end address> netmask <mask>
```

Configuring dynamic NAT from private inside addresses to a pool of public addresses:

```
Router(config)# ip nat inside source list <ACL> pool <pool>
```

The `type match-host` parameter can be appended to the `ip nat pool` statement so that the host portion of the address is reserved in the inside-to-outside translation.

NAT mappings can be viewed with `show ip nat translations [verbose]`.

Route maps can be used to determine how packets should be NATted out multiple outside interfaces:

```
ip nat inside source route-map ISP1_MAP pool ISP1
ip nat inside source route-map ISP2_MAP pool ISP2
!
route-map ISP1_MAP permit 10
 match ip address 1           ! Matches inside addresses
 match interface Serial0/0    ! Facing ISP1
!
route-map ISP2_MAP permit 10
 match ip address 1           ! Matches inside addresses
 match interface Serial0/1    ! Facing ISP2
```

PAT is enabled with the `overload` keyword:

```
Router(config)# ip nat inside source list 1 interface Serial0 overload
```

TCP load balancing pools are created with `type rotary`:

```
Router(config)# ip nat pool <name> 192.168.2.2 192.168.1.4 prefix-length 24 type rotary
```

Chapter 5: Introduction to IP Multicast Routing

The class D IP range (224.0.0.0/4) was reserved for multicast use. The address space is considered "flat" as multicast groups are not subnetted and aren't necessarily assigned in a hierarchical manner.

Multicast Ethernet MAC addresses are formed by concatenating `01-00-5e` and the lower 23 bits of the group ID.

IGMP

Internet Group Membership Protocol (IGMP) is used between routers and hosts receiving multicast traffic.

IGMP packets are restricted to the local data link.

There are three versions of IGMP (1, 2, and 3).

IGMPv2

IGMPv2 message types:

Membership Report - Indicates that a host wants to join a group

IGMPv1 Membership Report - Provided for backward compatibility

Leave Group - Indicates that a host has left a group

Membership reports are sent to the multicast group to ensure all routers and other group members on the segment receive the message. Leave messages are sent only to the *all routers* multicast group (224.0.0.2).

Routers can send two type of query to a subnet:

General - Polls to discover whether any group member are present, sent to *all hosts* (224.0.0.1)

Group-specific - Polls for members of a specific group, sent to the group address

General queries are sent every 60 seconds by default (modifiable with `ip igmp query-interval`).

The query contains a *Max Response Time* (default 10 seconds), indicating how long the router will wait to receive a response.

The router will stop forwarding traffic for the group if three consecutive queries go unanswered.

Group-specific queries are sent in response to receiving a leave message from a group member.

Only the active *querier* router sends queries. The querier for a subnet is the router with the lowest IP address.

If a non-querier does not hear queries from the querier within a period (default 120 seconds), it assumes the role. This timeout is configurable with `ip igmp query-timeout`.

IGMPv1

IGMPv1 is different from v2 in several ways:

IGMPv1 has no leave message

IGMPv1 has no group-specific query

- IGMPv1 hosts rely on a fixed Max Response Time of 10 seconds
- IGMPv1 relies on the multicast routing protocol to elect a querier

IGMPv2 hosts recognize IGMPv1 reports, and will respond to IGMPv1 queries with IGMPv1 reports.

An IGMPv2 router reverts to IGMPv1 operation for a group as long as an IGMPv1 member is present.

IGMPv3

IGMPv3 provides for source-specific group queries and filtering.

CGMP

Cisco Group Membership Protocol (CGMP) was used to communicate group membership to layer two switches to enable efficient forwarding of multicast traffic. Modern switches use IGMP snooping.

Multicast Routing

There are five IGPs to route multicast traffic:

- Distance Vector Multicast Routing Protocol (DVMRP)
- Multicast OSPF (MOSPF)
- Core-based Trees (CBT)
- Protocol Independent Multicast Dense Mode (PIM-DM)
- Protocol Independent Multicast Sparse Mode (PIM-SM)

Cisco IOS only fully supports PIM.

Multicast routes are expressed as (Source, Group) pairs.

Multicast packets should always travel away from the source. Multicast routing protocols use *reverse path forwarding* to ensure packets are not sent out interfaces which would be used to carry unicast traffic toward the source.

Dense Versus Sparse

A topology in which a large percentage of hosts receive multicast traffic is said to be *dense*. A *sparse* topology has comparatively few multicast destinations.

DVMRP, MOSPF, and PIM-DM are intended for dense mode deployment. CBT and PIM-SM are designed for sparse mode.

Implicit Versus Explicit Joins

A multicast tree is built using either *implicit* or *explicit joins*.

Implicit joins are used when a tree initially encompasses every router in the network, and is pruned back to include only segments with interested hosts. Traffic is reflooded and repruned at a regular interval.

Explicit joins are used to grow a multicast tree to reach all interested hosts.

DVMRP and PIM-DM use implicit joins; MOSPF, CBT, and PIM-SM use explicit joins.

Source-based Versus Shared Trees

Source-based trees are rooted at the multicast source and grown outward to the recipients. A separate tree is maintained for every source.

Shared trees are rooted at a shared, strategically positioned router called the *rendezvous point (RP)* and can be shared among multiple multicast groups.

Shared trees are represented with a single (*, G) route.

Although more efficient in routing, shared trees may not always provide an optimal path from a source to a recipient, as all traffic must flow through the RP.

Dense mode routing protocols use source-based trees; sparse mode protocols use shared trees.

Scoping

A multicast *scope* can be defined to limit the reach of multicast traffic.

TTL scoping sets TTL thresholds on routed interfaces. Multicast packets with a TTL lower than the threshold are not sent.

Administrative scoping sets individual boundaries for ranges with the reserved admin-scoped multicast address range (239.0.0.0/8).

DVMRP

[Skipped]

MOSPF

[Skipped]

CBT

[Skipped]

PIM-DM

PIM-DM uses five PIMv2 messages:

- Hello
- Join/Prune
- Graft
- Graft-Ack
- Assert

Hellos are transmitted every 30 seconds by default (configurable with `ip pim query-interval`) on every PIMv2 interface. Hellos contain a hold time set to 3.5 times the hello interval.

PIMv1 interfaces send queries instead of hellos.

Prune messages are multicast from leaf routers to 224.0.0.13.

Each multicast route has two timers: time in the table, and expiration timer. The route will be deleted if no multicast traffic is forwarded for the (S, G) pair before the expiration timer expires.

Pruned interfaces remain pruned for 210 seconds, after which traffic is flooded again. The downstream router must again request to be pruned from the tree.

A graft message is unicasted to an upstream neighbor to rejoin the tree. The neighbor transitions its downstream-facing interface to the forwarding state and replies with a graft-ack.

An upstream router will wait 3 seconds after receiving a prune message before actually pruning an interface. This gives other downstream neighbors on the same interface the opportunity to notice the prune message (multicasted to the segment) and send a *prune override*.

A prune override is accomplished by sending another join message to the upstream neighbor.

The highest PIM-DM router on a multiaccess segment becomes the *designated router (DR)*. The DR serves as the querier for IGMPv1 (IGMPv1 has no querier election process of its own).

Assert messages are used to designate the *forwarder* when multiple routers have a path from one segment to another.

The router advertising the most preferable unicast route (as determined by lowest admin distance/lowest metric/highest IP) to the source becomes the forwarder.

PIM-SM

PIM-SM uses seven PIMv2 messages:

- Hello
- Bootstrap
- Candidate RP Advertisement
- Join/Prune
- Assert
- Register
- Register-stop

The PIM-SM shared tree is rooted at the *rendezvous point (RP)* router, which can be designated in one of three ways:

- Static configuration on all routers
- Standards-based bootstrap protocol
- Cisco proprietary Auto-RP

Bootstrap Protocol

Candidate bootstrap routers (C-BSRs) and *candidate rendezvous points (C-RPs)* are administratively designated.

Bootstrap messages contain the originator's IP address and priority (0 through 255).

Bootstrap messages are multicast to *all PIM routers* (224.0.0.13) with a TTL of 1, and are flooded throughout the multicast domain to ensure all routers know of the BSR.

After the BSR has been established, C-RPs begin unicasting *candidate RP advertisement* messages to it, declaring their address, priority, and the groups they support.

The BSR collects all candidate RP advertisements into an *RP-set* and multicasts the RP-set to the multicast domain (224.0.0.13).

A router wishing to join the tree examines th RP-set and selects the C-RP with the lowest priority. If priorities are equal, the C-RP with the highest outcome of a hashing function is chosen. This ensures all routers select the same C-RP for a group as the RP.

Auto-RP

Auto-RP predates the bootstrap protocol and differs in several ways:

- Auto-RP is Cisco proprietary
- RP mapping agents are designated rather than elected from a set of candidates

RP mapping agents map groups to IPs instead of advertising an RP-set

Auto-RP uses *Cisco-RP-Announce* (224.0.1.39) and *Cisco-RP-Discovery* (224.0.1.40) instead of 224.0.0.13

Announcements are sent every 60 seconds.

Shared Trees

The multicast route entry for a shared tree is not source-specific; `(*, G)` is used instead of `(S, G)`.

Upon joining a group, a PIM-SM router takes several steps:

1. A (*, G) route is created for the interface(s) needing to receive traffic for the group (as indicated by IGMP join requests)
2. The group-to-RP mapping is consulted to find the RP for the group
3. The path to the RP is looked up in the unicast routing table
4. A Join/Prune message is sent to the RP requesting to join the group

Upstream routers receiving the join request extend the tree if they are already on it, or create and propagate their own join request toward the RP. This process repeats until the join request is intercepted by a router on the tree or the RP itself.

The RP will create a (*, G) route for the tree even if it does not yet know of a source for the group.

Join/Prune messages are resent every 60 seconds (configurable with `ip pim message-interval`) to keep the tree intact.

Prune overriding is used on multiaccess segments as with PIM-DM.

Source Registration and SPTs

The shared trees created by PIM-SM must be unidirectional so that RPF checking works. Multicast traffic from a source must reach the RP without relying on the group's shared tree.

When a source initially begins sending traffic to a group, its local router encapsulates the packets in a Register message and unicasts them to the RP. This ensures the RPF rule of the shared tree isn't violated.

When the RP receives the traffic encapsulated in a Register message, it forwards the traffic and creates an (S, G) route. The RP then builds a shortest-path tree to the source and sends a Register Stop message to the router which originated the Register messages.

PIM-SM routers can also build independent SPTs to the source if they realize they have a better path

than the shared tree.

By default a Cisco PIM-SM router will switch to a SPT whenever one becomes available. A threshold can be set (in kbps of multicast traffic to the group) with `ip pim spt-threshold` (default 0). A threshold of `infinity` can be specified to prevent ever switching to a SPT.

Chapter 6: Configuring and Troubleshooting IP Multicast Routing

The command `ip multicast-routing` in global configuration enables multicast routing and IGMP.

Configuring PIM-DM

PIM (all modes) is enabled per interface:

```
Router(config-if)# ip pim dense-mode
```

An IGP should be enabled on every PIM interface to avoid RPF failures.

An (S, G) route can only have one incoming interface, to ensure RPF works. In the event of equal-cost paths to the source, the router chooses the path to the neighbor with the higher IP address.

Configuring PIM-SM

Interface configuration is in line with PIM-DM:

```
Router(config-if)# ip pim sparse-mode
```

Static RP Configuration

An RP can be statically configured. It should be configured on all PIM routers in the domain, including the RP itself.

```
Router(config)# ip pim rp-address <IP>
```

It is recommended to use a loopback interface as the RP address. This interface does not have to run PIM.

An access list can optionally be appended to designate an RP only for particular groups:

```
Router(config)# access-list 10 permit 239.0.16.78 0.0.0.0
```

```
Router(config)# ip pim rp-address <IP> 10
```

Auto-RP Configuration

Automatic RP discovery (through Auto-RP or bootstrap) is recommended for larger multicast domains to provide better management and redundancy.

Auto-RP configuration requires two steps: configuring candidate RPs (C-RPs) and configuring mapping agents.

C-RPs are configured with the `ip pim send-rp-announce` command:

```
Router(config)# ip pim send-rp-announce <interface> scope <TTL> [interval <seconds>]
```

The RP address is taken from the specified interface, and advertisements receive the specified TTL.

RP-announce messages are multicast to 224.0.1.39 every 60 seconds by default.

Mapping agents are configured with the `ip pim send-rp-discovery` command:

```
Router(config)# ip pim send-rp-discovery <interface> scope <TTL>
```

The interface specified must have PIM-SM configured.

The mapping agent listens for RP-announce messages and selects the RP for each group. The RPs are advertised in RP-discovery messages multicasted to 224.0.1.40 every 60 seconds.

The mapping agents chooses the RP for each group by highest IP address.

To specify only particular groups for which a router is a C-RP, append a `group-list` specifying an ACL which matches the appropriate multicast groups:

```
Router(config)# ip pim send-rp-announce <interface> scope <TTL> group-list <ACL>
```

Active RPs can be viewed with `show ip pim rp [mapping]`.

Sparse-Dense Mode

Sparse-dense mode can be configured to facilitate PIM dense mode initially, switching to sparse mode after an RP is known.

Interface configuration:

```
Router(config-if)# ip pim sparse-dense-mode
```

Bootstrap Protocol Configuration

Configuring candidate BSRs:

```
Router(config)# ip pim bsr-candidate <interface> [<hash mask length>] [<priority>]
```

By default BSRs have a hash mask length and priority of 0.

Configuring candidate RPs:

```
Router(config)# ip pim rp-candidate <interface>
```

`show ip pim bsr-router` shows the active BSR.

Candidate RPs unicast their Candidate-RP-Advertisement messages to the BSR. The BSR condenses these into an RP-Set which is included in Bootstrap messages.

As with Auto-RP, a `group-list` can be appended to the `rp-candidate` statement to determine the groups for which the router can become an RP.

Multicast Load Sharing

A tunnel can be implemented to achieve load sharing across multiple paths without breaking the RPF rule.

Troubleshooting

`mrinfo` can be used to examine a router's multicast connections, or those of a neighbor:

```
R1# mrinfo 2.2.2.2
2.2.2.2 [version  12.4] [flags: PMA]:
  10.0.0.2 -> 10.0.0.1 [1/0/pim/querier]
  10.0.0.9 -> 10.0.0.10 [1/0/pim]
```

`mtrace <source> <destination>` displays an RPF path trace:

```
R1# mtrace 172.16.40.1 172.16.20.1
Type escape sequence to abort.
Mtrace from 172.16.40.1 to 172.16.20.1 via RPF
From source (?) to destination (?)
Querying full reverse path...
 0  172.16.20.1
-1  10.0.0.14 PIM  [172.16.40.0/24]
-2  10.0.0.13 PIM  [172.16.40.0/24]
-3  10.0.0.5 PIM  [172.16.40.0/24]
-4  172.16.40.1
```

`mstat` is an advanced version of `mtrace` and provides a trace as well as path statistics.

Chapter 7: Large-Scale IP Multicast Routing

Multicast Scoping

Two methods of scoping a multicast domain exist:

- TTL scoping
- Administrative scoping

TTL scoping configured at the interface:

```
Router(config-if)# ip multicast ttl-threshold <minimum TTL>
```

Multicast packets with a TTL less than the minimum TTL are dropped.

Administrative scoping references an access list to permit or deny multicast packets per group:

```
Router(config-if)# ip multicast boundary <ACL>
```

Multicasting Across Non-multicast Domains

Point-to-point tunnels can be employed to transport multicast traffic across a network lacking multicast support.

Inter-AS Multicasting

Multiprotocol BGP (MBGP) is an extension of BGP which supports multicast routes.

Multicast Source Discovery Protocol (MSDP) is used to share source information among RPs in different autonomous systems.

MBGP

MBGP introduces two additional attributes:

MP_REACH_NLRI - Advertises multicast routes

MP_UNREACH_NLRI - Withdraws multicast routes

Both attributes are optional, nontransitive.

MBGP is configured by defining a multicast address family with `address-family ipv4 multicast`:

```
router bgp <AS>
 no synchronization
 network <network>
 neighbor <neighbor>
 !
 address-family ipv4 multicast
 network <network>
 neighbor <neighbor> activate
 exit-address-family
```

The networks specified under the multicast address family advertise multicast sources.

MSDP

MSDP uses explicitly configured point-to-point peer connections on TCP port 639.

When a PIM RP register a multicast source, it sends a *Source Active (SA)* message to all of its MSDP peers. SAs are flooded through the MSDP domain.

The next-hop neighbor for unicast traffic toward the advertised source (as determined by MBGP or BGP) is the *RPF peer*. To prevent looping, all SAs originating on interfaces not facing the RPF peer are dropped instead of flooded.

If an RP receiving an SA determines it has members for the advertised group, it sends an (S, G) join to the source.

RPs can optionally cache SAs, minimizing the delay observed by a host wishing to join an MSDP-advertised group. Caching (disabled by default) can be enabled in IOS with `ip msdp cache-sa-state`.

Join latency can also be reduced by implementing *SA Requests* with `ip msdp sa-request`. An RP receiving a join message for a group it does not have will then issue an SA Request to its peers which are caching. If an *SA Response* is received, it is not flooded to other neighbors.

MSDP messages:

1. Source Active
2. Source Active Request
3. Source Active Response
4. Keepalive
5. Notification (error)

Defining an MSDP peer:

```
Router(config)# ip msdp peer <address> [connect-source <interface>]
```

MSDP peers are inspected with `show ip msdp peer`.

Setting a peer description:

```
Router(config)# ip msdp description <address> <string>
```

Setting the TTL of messages sent to the peer:

```
Router(config)# ip msdp ttl-threshold <address> <TTL>
```

Enable SA caching:

```
Router(config)# ip msdp cache-sa-state [list <ACL>]
```

Initiate SA Requests to caching neighbors:

```
Router(config)# ip msdp sa-request <address>
```

Filter SA Requests:

```
Router(config)# ip msdp filter-sa-request <address> list <ACL>
```

Filter SA messages:

```
Router(config)# ip msdp sa-filter <address> {in | out} list <ACL>
```

MSDP mesh groups can be used to reduce flooding between MSDP peers connected in a full mesh:

```
Router(config)# ip msdp mesh-group <name> <peer 1>
Router(config)# ip msdp mesh-group <name> <peer 2>
Router(config)# ip msdp mesh-group <name> ...
```

The RPs in a mesh group can emulate an *anycast RP* by all advertising RP capability (through Auto-RP or Bootstrap) from the same IP address configured on a loopback interface. MSDP sessions are source from a separate, unique loopback on each RP.

To set the uniquely addressed interface as the originator ID:

```
Router(config)# ip msdp originator-id <interface>
```

MSDP's reliance on BGP for RPF checking can be disabled by specifying a *default peer* from which all SA messages are accepted:

```
Router(config)# ip msdp default-peer <address>
```

If multiple default peers are configured, SA messages are only accepted from one of them; the remaining peers provide for failover.

Chapter 9: Router Management

SNMP

Simple Network Management Protocol (SNMP) consists of managers and agents. Managers collect data provided by agents.

SNMP message types:

- Get
- Set

Trap

Parameter are defined in a *Management Information Base (MIB)*.

Configuring an SNMP community:

```
Router(config)# snmp-server community <string> [view <name>] [ro | rw] [ACL]
```

An ACL, if appended, matches hosts which may query the device. `ro` or `rw` designates the community as read-only or read-write, respectively.

Views can be defined to limit access to only parts of the MIB tree:

```
Router(config)# snmp-server view <name> <OID tree> {included | excluded}
```

Configuring a host to receive traps:

```
Router(config)# snmp-server host <address> [version {1 | 2c}] <community>
```

To enable or disable specific trap types:

```
Router(config)# [no] snmp-server enable traps <type> [<option>]
```

RMON

The *Remote Monitoring (RMON)* engine on a router polls SNMP MIB variable locally. An SNMP trap is only sent when a variable crosses its rising or falling threshold.

Configuring an RMON alarm:

```
Router(config)# rmon alarm <number> <variable> <interval> {delta | absolute}
 rising-threshold <value> [<event>] falling-threshold <value> [<event>]
```

Configuring an event to be triggered by an alarm:

```
Router(config)# rmon event <number> [log] [trap <community>] [description <string>]
```

Logging

Enabling log timestamps:

```
Router(config)# service timestamps log {uptime | datetime}
```

Logging levels:

- **0** - Emergencies
- **1** - Alerts
- **2** - Critical
- **3** - Errors
- **4** - Warnings
- **5** - Notifications
- **6** - Informational
- **7** - Debugging

Configuring syslog:

```
Router(config)# logging <server>
```

NTP

The *stratum* level defines the degree of reliability associated with a timing source.

NTP is efficient; one packet per minute allows two devices to synchronize within 10ms.

Configuring an NTP association:

```
Router(config)# ntp {server | peer} <address> [version <number>] [key <number>]
 [source <interface>] [prefer]
```

Server associations are configured when synchronization is one-way (client to server). Peer associations allow for bidirectional synchronization between two sources.

To restrict access to a router's NTP service:

```
Router(config)# ntp access-group {query-only | serve-only | serve | peer} <ACL>
```

To designate a router as a master time source (which does not reference an external clock):

```
Router(config)# ntp master [<stratum>]
```

The default stratum for a master is 8.

Enabling NTP authentication:

```
Router(config)# ntp authenticate
Router(config)# ntp authentication-key <number> md5 <string>
Router(config)# ntp trusted-key <number>
Router(config)# ntp server <address> key <number>
```

NTP status is verified with `show ntp status` and `show ntp associations`.

Accounting

IP accounting provides basic accounting services for packets traversing a router.

IP accounting is enabled at the interface:

```
Router(config)# ip accounting
```

Statistics are monitored with `show ip accounting`.

NetFlow accounting provides more granular, easily exported detail.

NetFlow switching is configured at the interface:

```
Router(config-if)# ip route-cache flow
```

Defining export to a NetFlow collector:

```
Router(config)# ip flow-export destination <address> <port>
Router(config)# ip flow-export [version <version>]
```

`show ip flow export` displays NetFlow export information.

Flows can be aggregated by AS numbers, prefixes, or port numbers with `ip flow-aggregation <...>`.

TACACS+

Terminal Access Controller Access Control System Plus (TACACS+) provides centralized control over authentication, authorization, and accounting (AAA).

`aaa new-model` is necessary to enable AAA on the router before configuring TACACS+.

Configuring TACACS+ authentication:

```
Router(config)# aaa authentication login {default | <name>} group <type>
```

If a non-default name is specified, the authentication method must be explicitly applied to lines:

```
Router(config)# line vty 0 15
Router(config-line)# login authentication <name>
```

Specifying a TACACS+ server and a shared encryption key:

```
Router(config)# tacacs-server host <address>
Router(config)# tacacs-server key <string>
```

Configuring TACACS+ authorization:

```
Router(config)# aaa authorization {network | exec | commands <level>} {default |
 <name>} [<method> ...]
```

Configuring TACACS+ accounting:

```
Router(config)# aaa accounting {system | network | exec | connection | commands
 <level>} {default | <name>} {start-stop | wait-start | stop-only | none}
 [<method> ...]
```

RADIUS

Remote Access Dial-In Service (RADIUS) provides many of the same services as TACACS+ but does not allow for restricting the use of individual commands.

RADIUS is enabled ver similarly to TACACS+, substituting 'radius' for 'tacacs' or 'tacacs+' where appropriate.