BGP Case Studies 1

The BGP, which RFC 1771 defines, allows you to create loop-free interdomain routing between autonomous systems (ASs). An AS is a set of routers under a single technical administration. Routers in an AS can use multiple Interior Gateway Protocols (IGPs) to exchange routing information inside the AS. The routers can use an exterior gateway protocol to route packets outside the AS.

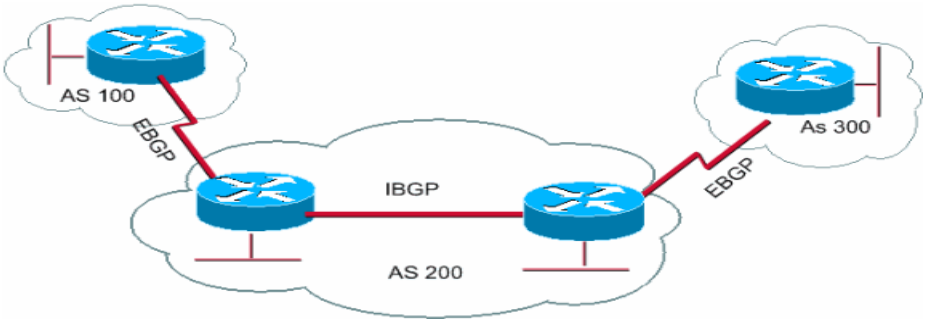
How Does BGP Work? BGP uses TCP as the transport protocol, on port 179. Two BGP routers form a TCP connection between one another. These routers are peer routers. The peer routers exchange messages to open and confirm the connection parameters.BGP routers exchange network reachability information. This information is mainly an indication of the full paths that a route must take in order to reach the destination network. The paths are BGP AS numbers. This information helps in the construction of a graph of ASs that are loop-free. The graph also shows where to apply routing policies in order to enforce some restrictions on the routing behavior. Any two routers that form a TCP connection in order to exchange BGP routing information are "peers" or "neighbors". BGP peers initially exchange the full BGP routing tables. After this exchange, the peers send incremental updates as the routing table changes. BGP keeps a version number of the BGP table. The version number is the same for all the BGP peers. The version number changes whenever BGP updates the table with routing information changes. The send of keepalive packets ensures that the connection between the BGP peers is alive. Notification packets go out in response to errors or special conditions.

eBGP and iBGP

If an AS has multiple BGP speakers, the AS can serve as a transit service for other ASs. As the diagram in this section shows, AS200 is a transit AS for AS100 and AS300. In order to send the information to external ASs, there must be an assurance of the reachability for networks. In order to assure network reachability, these processes take place:

- Internal BGP (iBGP) peering between routers inside an AS
- Redistribution of BGP information to IGPs that run in the AS

When BGP runs between routers that belong to two different ASs, this is called exterior BGP (eBGP). When BGP runs between routers in the same AS, this is called iBGP.



Enable BGP Routing

Complete these steps in order to enable and configure BGP.

Assume that you want to have two routers, RTA and RTB, talk via BGP. In the first example, RTA and RTB are in different ASs. In the second example, both routers belong to the same AS.

1. Define the router process and the AS number to which the routers belong.
   Issue this command to enable BGP on a router:

   router bgp *autonomous-system*

RTA#
router bgp 100

RTB#
router bgp 200

These statements indicate that RTA runs BGP and belongs to AS100. RTB runs BGP and belongs to AS200.

1. Define BGP neighbors.

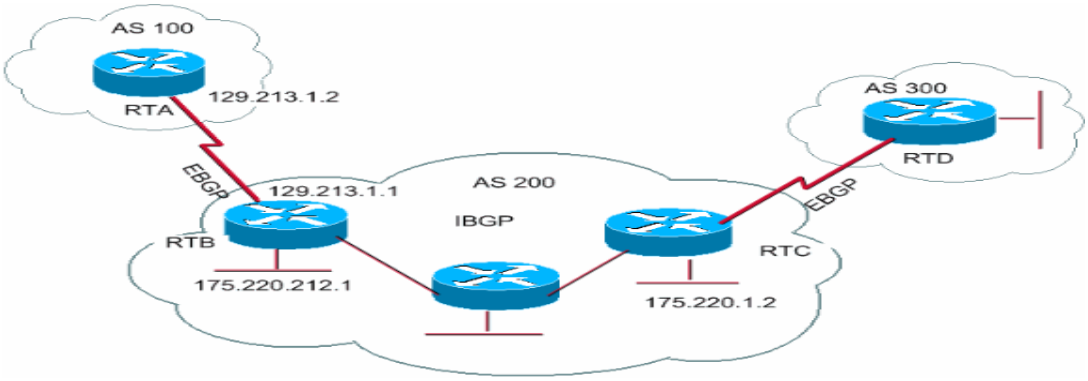
   The BGP neighbor formation indicates the routers that attempt to talk via BGP. The section Form BGP Neighbors explains this process.

Form BGP Neighbors

Two BGP routers become neighbors after the routers establish a TCP connection between each other. The TCP connection is essential in order for the two peer routers to start the exchange of routing updates. After the TCP connection is up, the routers send open messages in order to exchange values. The values that the routers exchange include the AS number, the BGP version that the routers run, the BGP router ID, and the keepalive hold time. After the confirmation and acceptance of these values, establishment of the neighbor connection occurs. Any state other than Established is an indication that the two routers did not become neighbors and that the routers cannot exchange BGP updates.

Issue this **neighbor** command to establish a TCP connection: **neighbor** *ip-address* **remote-as** *number*
The ***number*** in the command is the AS number of the router to which you want to connect with BGP. The ***ip-address*** is the next hop address with direct connection for eBGP. For iBGP, ***ip-address*** is any IP address on the other router. The two IP addresses that you use in the **neighbor** command of the peer routers *must* be able to reach one another. One way to verify reachability is an extended ping between the two IP addresses. The extended ping forces the pinging router to use as source the IP address that the **neighbor** command specifies. The router must use this address rather than the IP address of the interface from which the packet goes.If there are any BGP configuration changes, you *must* reset the neighbor connection to allow the new parameters to take effect.

- **clear ip bgp** *address*
  Note: The ***address*** is the neighbor address.
- **clear ip bgp ***
  This command clears all neighbor connections.
  By default, BGP sessions begin with the use of BGP version 4 and negotiate downward to earlier versions, if necessary. You can prevent negotiations and force the BGP version that the routers use to communicate with a neighbor. Issue this command in router configuration mode: **neighbor {***ip address* **|** *peer-group-name***} version** *value*
  Here is an example of the **neighbor** command configuration:



RTA#

```
router bgp 100
neighbor 129.213.1.1 remote-as 200

RTB#
router bgp 200
neighbor 129.213.1.2 remote-as 100
neighbor 175.220.1.2 remote-as 200

RTC#
router bgp 200
neighbor 175.220.212.1 remote-as 200
```

In this example, RTA and RTB run eBGP. RTB and RTC run iBGP. The remote AS number points to either an external or an internal AS, which indicates either eBGP or iBGP. Also, the eBGP peers have direct connection, but the iBGP peers do not have direct connection. iBGP routers do not need to have direct connection. But, there must be some IGP that runs and allows the two neighbors to reach one another.

This section provides an example of the information that the **show ip bgp neighbors** command displays.

Note: Pay special attention to the BGP state. Anything other than the state Established indicates that the peers are not up.

Note: Also, notice these items:

- The BGP version, which is 4
- The remote router ID

  This number is the highest IP address on the router or the highest loopback interface, if existent.

- The table version

  The table version provides the state of the table. Any time that new information comes in, the table increases the version. A version that continues to increment indicates that there is some route flap that causes the continuous update of routes.

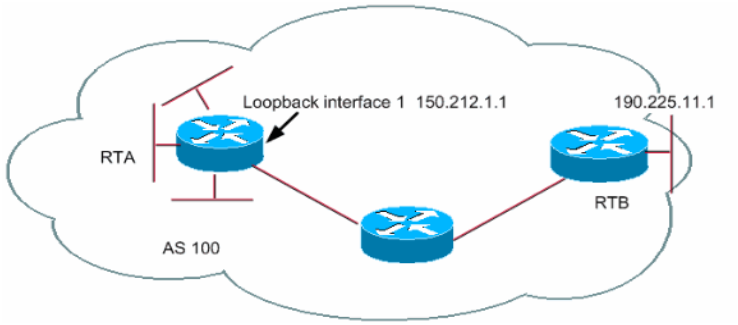
```
# show ip bgp neighbors
   BGP neighbor is 129.213.1.1, remote AS 200, external link
   BGP version 4, remote router ID 175.220.12.1
   BGP state = Established, table version = 3, up for 0:10:59
   Last read 0:00:29, hold time is 180, keepalive interval is 60 seconds
   Minimum time between advertisement runs is 30 seconds
   Received 2828 messages, 0 notifications, 0 in queue
   Sent 2826 messages, 0 notifications, 0 in queue
   Connections established 11; dropped 10
```

BGP and Loopback Interfaces

The use of a loopback interface to define neighbors is common with iBGP, but is not common with eBGP. Normally, you use the loopback interface to make sure that the IP address of the neighbor stays up and is independent of hardware that functions properly. In the case of eBGP, peer routers frequently have direct connection, and loopback does not apply. If you use the IP address of a loopback interface in the **neighbor** command, you need some extra configuration on the neighbor router. The neighbor router needs to inform BGP of the use of a loopback interface rather than a physical interface to initiate the BGP neighbor TCP connection. In order to indicate a loopback interface, issue this command:

neighbor *ip-address* **update-source** *interface*

This example illustrates the use of this command:




```
RTA#
    router bgp 100
    neighbor 190.225.11.1 remote-as 100
    neighbor 190.225.11.1 update-source loopback 1
RTB#
    router bgp 100
    neighbor 150.212.1.1 remote-as 100
```

In this example, RTA and RTB run iBGP inside AS100. In the **neighbor** command, RTB uses the loopback interface of RTA, 150.212.1.1. In this case, RTA must force BGP to use the loopback IP address as the source in the TCP neighbor connection. In order to force this action, RTA adds **update-source** *interface-typeinterface-number* so that the command is **neighbor 190.225.11.1 update-source loopback 1**. This statement forces BGP to use the IP address of the loopback interface when BGP talks to neighbor 190.225.11.1.

Note: RTA has used the physical interface IP address of RTB, 190.225.11.1, as a neighbor. Use of this IP address is why RTB does not need any special configuration. Refer to Sample Configuration for iBGP and eBGP With or Without a Loopback Address for a complete network scenario sample configuration.

eBGP Multihop

In some cases, a Cisco router can run eBGP with a third-party router that does not allow direct connection of the two external peers. To achieve the connection, you can use eBGP multihop. The eBGP multihop allows a neighbor connection between two external peers that do not have direct connection. The multihop is only for eBGP and not for iBGP. This example illustrates eBGP multihop:

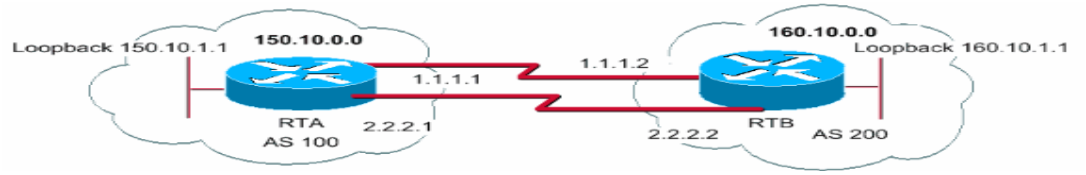

```
RTA#
router bgp 100
neighbor 180.225.11.1 remote-as 300
neighbor 180.225.11.1 ebgp-multihop
RTB#
router bgp 300
```

neighbor 129.213.1.2 remote-as 100

RTA indicates an external neighbor that does not have direct connection. RTA needs to indicate its use of the **neighbor ebgp-multihop** command. On the other hand, RTB indicates a neighbor that has direct connection, which is 129.213.1.2. Because of this direct connection, RTB does not need the **neighbor ebgp-multihop** command. You should also configure an IGP or static routing to allow the neighbors without connection to reach each other.

The example in the eBGP Multihop (Load Balancing) section shows how to achieve load balancing with BGP in a case where you have eBGP over parallel lines.

eBGP Multihop (Load Balancing)



RTA#
int loopback 0
ip address 150.10.1.1 255.255.255.0
router bgp 100
neighbor 160.10.1.1 remote-as 200
neighbor 160.10.1.1 ebgp-multihop
neighbor 160.10.1.1 update-source loopback 0
network 150.10.0.0

ip route 160.10.0.0 255.255.0.0 1.1.1.2
ip route 160.10.0.0 255.255.0.0 2.2.2.2
RTB#
int loopback 0
ip address 160.10.1.1 255.255.255.0
router bgp 200
neighbor 150.10.1.1 remote-as 100
neighbor 150.10.1.1 update-source loopback 0
neighbor 150.10.1.1 ebgp-multihop
network 160.10.0.0

ip route 150.10.0.0 255.255.0.0 1.1.1.1
ip route 150.10.0.0 255.255.0.0 2.2.2.1

This example illustrates the use of loopback interfaces, **update-source**, and **ebgp-multihop**. The example is a workaround in order to achieve load balancing between two eBGP speakers over parallel serial lines. In normal situations, BGP picks one of the lines on which to send packets, and load balancing does not happen. With the introduction of loopback interfaces, the next hop for eBGP is the loopback interface. You use static routes, or an IGP, to introduce two equal-cost paths to reach the destination. RTA has two choices to reach next hop 160.10.1.1: one path via 1.1.1.2 and the other path via 2.2.2.2. RTB has the same choices.

Route Maps

There is heavy use of route maps with BGP. In the BGP context, the route map is a method to control and modify routing information. The control and modification of routing information occurs through the definition of conditions for route redistribution from one routing protocol to another. Or the control of routing information can occur at injection in and out of BGP. The format of the route map follows:

route-map *map-tag* **[[permit | deny] | [***sequence-number***]]**

The map tag is simply a name that you give to the route map. You can define multiple instances of the same route map, or the same name tag. The sequence number is simply an indication of the position that a new route map is to have in the list of route maps that you have already configured with the same name.

In this example, there are two instances of the route map defined, with the name MYMAP. The first instance has a sequence number of 10, and the second has a sequence number of 20.

- **route-map MYMAP permit 10** (The first set of conditions goes here.)
- **route-map MYMAP permit 20** (The second set of conditions goes here.)

When you apply route map MYMAP to incoming or outgoing routes, the first set of conditions are applied via instance 10. If the first set of conditions is not met, you proceed to a higher instance of the route map.

match and set Configuration Commands

Each route map consists of a list of **match** and **set** configuration commands. The match specifies a **match** criteria, and set specifies a **set** action if the criteria that the **match** command enforces are met.

For example, you can define a route map that checks outgoing updates. If there is a match for IP address 1.1.1.1, the metric for that update is set to 5. These commands illustrate the example:

match ip address *1.1.1.1*
set metric *5*

Now, if the match criteria are met and you have a **permit**, there is a redistribution or control of the routes, as the set action specifies. You break out of the list.

If the match criteria are met and you have a **deny**, there is no redistribution or control of the route. You break out of the list.

If the match criteria are not met and you have a **permit or deny**, the next instance of the route map is checked. For example, instance 20 is checked. This next-instance check continues until you either break out or finish all the instances of the route map. If you finish the list without a match, the route is **not accepted nor forwarded**.

In Cisco IOS® Software releases earlier than Cisco IOS Software Release 11.2, when you use route maps to filter BGP updates rather than redistribute between protocols, you *cannot* filter on the inbound when you use a **match** command on the IP address. A filter on the outbound is acceptable. Cisco IOS Software Release 11.2 and later releases do not have this restriction.

The related commands for **match** are:

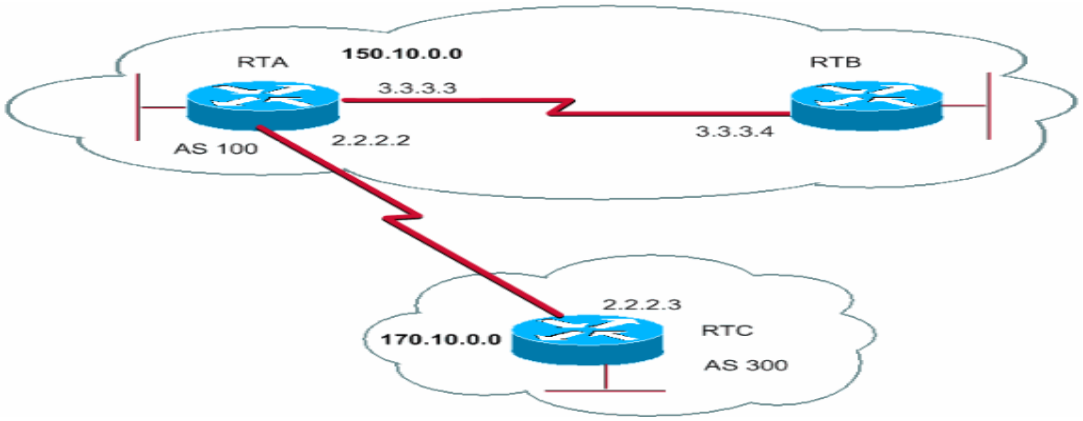
- **match as-path**
- **match community**
- **match clns**
- **match interface**
- **match ip address**
- **match ip next-hop**
- **match ip route-source**
- **match metric**
- **match route-type**

- **match tag**

The related commands for **set** are:

- **set as-path**
- **set clns**
- **set automatic-tag**
- **set community**
- **set interface**
- **set default interface**
- **set ip default next-hop**
- **set level**
- **set local-preference**
- **set metric**
- **set metric-type**
- **set next-hop**
- **set origin**
- **set tag**
- **set weight**

Look at some route map examples:



Example 1

Assume that RTA and RTB run Routing Information Protocol (RIP), and RTA and RTC run BGP. RTA gets updates via BGP and redistributes the updates to RIP. Suppose that RTA wants to redistribute to RTB routes about 170.10.0.0 with a metric of 2 and all other routes with a metric of 5. In this case, you can use this configuration:

```
RTA#
router rip
network 3.0.0.0
network 2.0.0.0
network 150.10.0.0
passive-interface Serial0
redistribute bgp 100 route-map SETMETRIC

router bgp 100
neighbor 2.2.2.3 remote-as 300
network 150.10.0.0

route-map SETMETRIC permit 10
```

```
match ip-address 1
set metric 2

route-map SETMETRIC permit 20
set metric 5

access-list 1 permit 170.10.0.0 0.0.255.255
```

In this example, if a route matches the IP address 170.10.0.0, the route has a metric of 2. Then, you break out of the route map list. If there is no match, you proceed down the route map list, which indicates setting everything else to metric 5.

Note: Always ask the question "What happens to routes that do not match any of the match statements?" These routes drop, by default.

Example 2

Suppose that, in Example 1, you do not want AS100 to accept updates about 170.10.0.0. You cannot apply route maps on the inbound when you match with an IP address as the basis. Therefore, you must use an outbound route map on RTC:

```
RTC#

router bgp 300
network 170.10.0.0
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 route-map STOPUPDATES out

route-map STOPUPDATES permit 10
match ip address 1

access-list 1 deny 170.10.0.0 0.0.255.255
access-list 1 permit 0.0.0.0 255.255.255.255
```

Now that you feel more comfortable with how to start BGP and how to define a neighbor, look at how to start the exchange of network information.

There are multiple ways to send network information with use of BGP. These sections go through the methods one by one:

- network Command
- Redistribution
- Static Routes and Redistribution

network Command

The format of the **network** command is:

network *network-number* **[mask** *network-mask***]**

The **network** command controls the networks that originate from this box. This concept is different than the familiar configuration with Interior Gateway Routing Protocol (IGRP) and RIP. With this command, you do not try to run BGP on a certain interface. Instead, you try to indicate to BGP what networks BGP should originate from this box. The command uses a mask portion because BGP version 4 (BGP4) can handle subnetting and supernetting. A maximum of 200 entries of the **network** command are acceptable.

The **network** command works if the router knows the network that you attempt to advertise, whether connected, static, or learned dynamically.

An example of the **network** command is:

```
RTA#
router bgp 1
network 192.213.0.0 mask 255.255.0.0
ip route 192.213.0.0 255.255.0.0 null 0
```

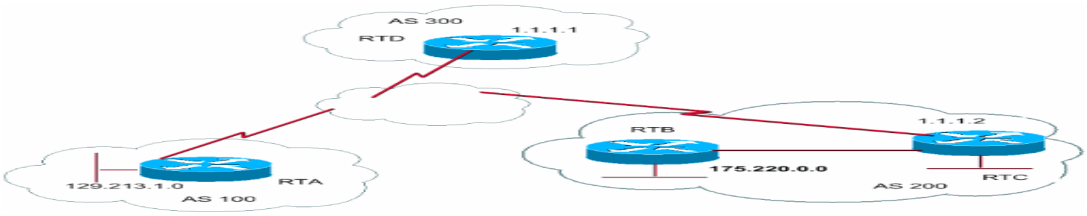
This example indicates that router A generates a network entry for 192.213.0.0/16. The /16 indicates that you use a supernet of the class C address and you advertise the first two octets, or first 16 bits.

Note: You need the static route to get the router to generate 192.213.0.0 because the static route puts a matching entry in the routing table.

Redistribution

The **network** command is one way to advertise your networks via BGP. Another way is to redistribute your IGP into BGP. Your IGP can be IGRP, Open Shortest Path First (OSPF) protocol, RIP, Enhanced Interior Gateway Routing Protocol (EIGRP), or another protocol. This redistribution can seem scary because now you dump all your internal routes into BGP; some of these routes can have been learned via BGP and you do not need to send them out again. Apply careful filtering to make sure that you send to the Internet-only routes that you want to advertise and not to all the routes that you have. Here is an example:

RTA announces 129.213.1.0 and RTC announces 175.220.0.0. Look at the RTC configuration:



If you issue the **network** command, you have:

```
RTC#
router eigrp 10
network 175.220.0.0
redistribute bgp 200
default-metric 1000 100 250 100 1500

router bgp 200
neighbor 1.1.1.1 remote-as 300
network 175.220.0.0 mask 255.255.0.0
```

!--- This limits the networks that your AS originates to 175.220.0.0.

If you use redistribution instead, you have:

```
RTC#
router eigrp 10
network 175.220.0.0
redistribute bgp 200
```

```
default-metric 1000 100 250 100 1500

router bgp 200
neighbor 1.1.1.1 remote-as 300
redistribute eigrp 10

!--- EIGRP injects 129.213.1.0 again into BGP.
```

This redistribution causes the origination of 129.213.1.0 by your AS. You are not the source of 129.213.1.0; AS100 is the source. So you have to use filters to prevent the source out of that network by your AS. The correct configuration is:

```
RTC#
router eigrp 10
network 175.220.0.0
redistribute bgp 200
default-metric 1000 100 250 100 1500

router bgp 200
neighbor 1.1.1.1 remote-as 300
neighbor 1.1.1.1 distribute-list 1 out
redistribute eigrp 10

access-list 1 permit 175.220.0.0 0.0.255.255
```

You use the **access-list** command to control the networks that originate from AS200.

Redistribution of OSPF into BGP is slightly different than redistribution for other IGPs. The simple issue of **redistribute ospf 1** under **router bgp** does not work. Specific keywords such as **internal**, **external**, and **nssa-external** are necessary to redistribute respective routes. Refer to Understanding Redistribution of OSPF Routes into BGP for more details.

Static Routes and Redistribution

You can always use static routes to originate a network or a subnet. The only difference is that BGP considers these routes to have an origin that is incomplete, or unknown. You can accomplish the same result that the example in the Redistribution section accomplished with this:

```
RTC#
router eigrp 10
network 175.220.0.0
redistribute bgp 200
default-metric 1000 100 250 100 1500

router bgp 200
neighbor 1.1.1.1 remote-as 300
redistribute static
...
ip route 175.220.0.0 255.255.255.0 null0
....
```

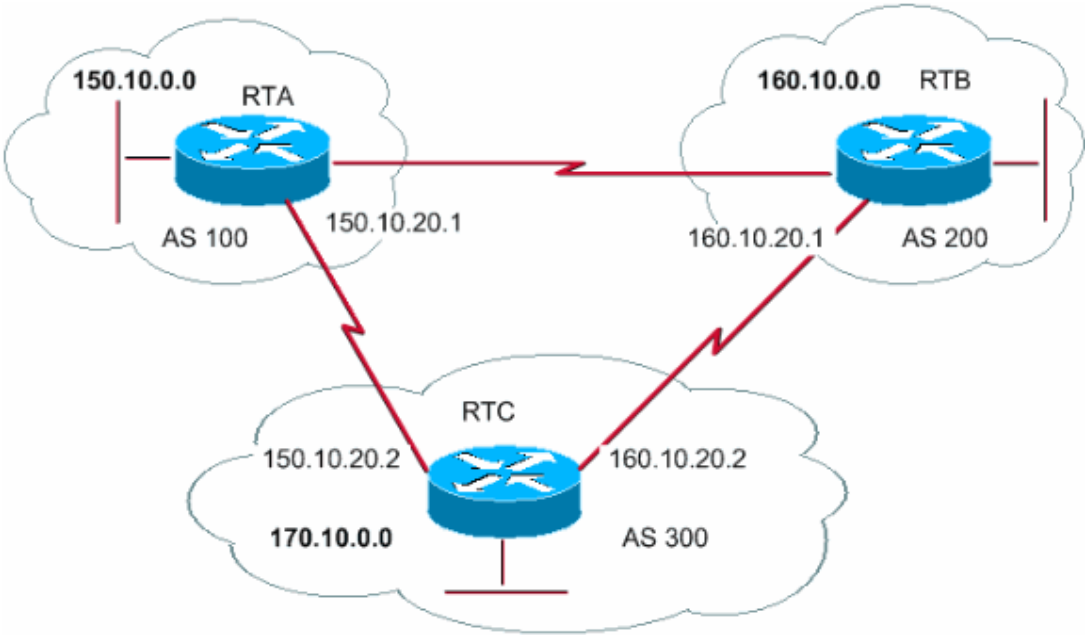
The **null0** interface means disregard the packet. So if you get the packet and there is a more specific match than 175.220.0.0, which exists, the router sends the packet to the specific match. Otherwise, the router disregards the packet. This method is a nice way to advertise a supernet.

This document has discussed how you can use different methods to originate routes out of your AS. Remember that these routes are generated in addition to other BGP routes that BGP has learned via

neighbors, either internal or external. BGP passes on information that BGP learns from one peer to other peers. The difference is that routes that generate from the **network** command, redistribution, or static indicate your AS as the origin of these networks.

Redistribution is always the method for injection of BGP into IGP.

Here is an example:



RTA#
router bgp 100
neighbor 150.10.20.2 remote-as 300
network 150.10.0.0

RTB#
router bgp 200
neighbor 160.10.20.2 remote-as 300
network 160.10.0.0

RTC#
router bgp 300
neighbor 150.10.20.1 remote-as 100
neighbor 160.10.20.1 remote-as 200
network 170.10.00

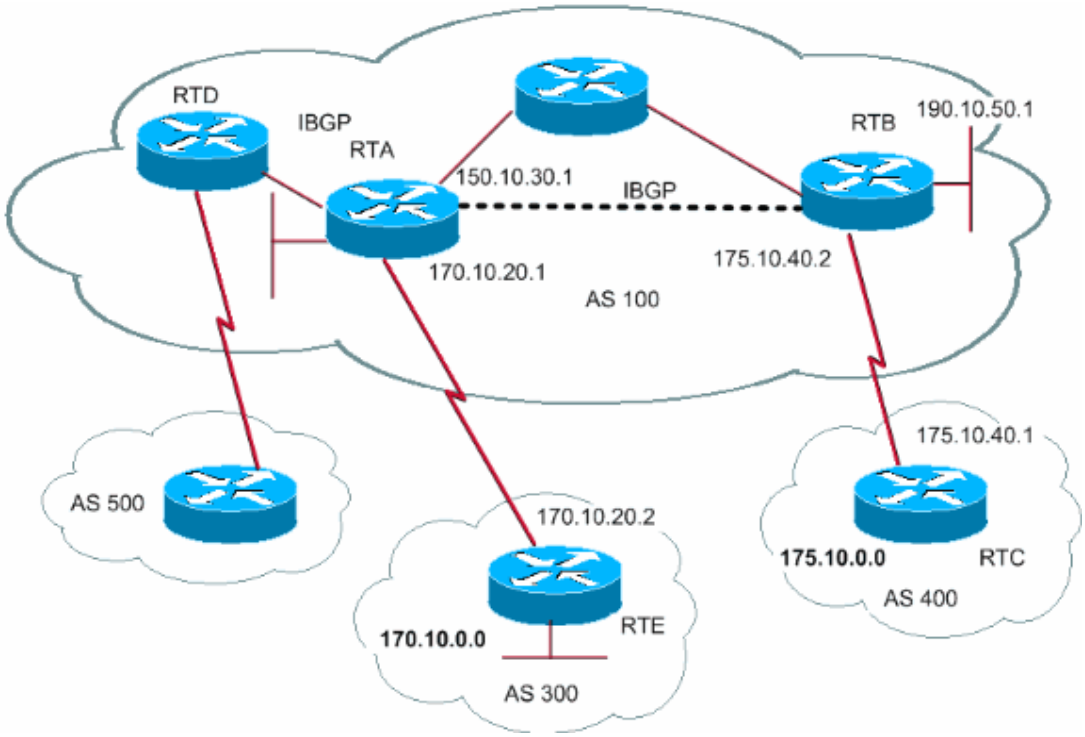
Note: You do not need network 150.10.0.0 or network 160.10.0.0 in RTC unless you want RTC to generate these networks as well as pass on these networks as they come in from AS100 and AS200. Again, the difference is that the **network** command adds an extra advertisement for these same networks, which indicates that AS300 is also an origin for these routes.

Note: Remember that BGP does not accept updates that have originated from its own AS. This refusal ensures a loop-free interdomain topology.

For example, assume that AS200, from the example in this section, has a direct BGP connection into AS100. RTA generates a route 150.10.0.0 and sends the route to AS300. Then, RTC passes this route to AS200 and keeps the origin as AS100. RTB passes 150.10.0.0 to AS100 with the origin still AS100. RTA notices that the update has originated from its own AS and ignores the update.

iBGP

You use iBGP if an AS wants to act as a transit system to other ASs. Is it true that you can do the same thing by learning via eBGP, redistributing into IGP, and then redistributing again into another AS? Yes, but iBGP offers more flexibility and more efficient ways to exchange information within an AS. For example, iBGP provides ways to control the best exit point out of the AS with use of local preference. The section Local Preference Attribute provides more information about local preference.



RTA#
router bgp 100
neighbor 190.10.50.1 remote-as 100
neighbor 170.10.20.2 remote-as 300
network 150.10.0.0

RTB#
router bgp 100
neighbor 150.10.30.1 remote-as 100
neighbor 175.10.40.1 remote-as 400
network 190.10.50.0

RTC#
router bgp 400
neighbor 175.10.40.2 remote-as 100
network 175.10.0.0

Note: Remember that when a BGP speaker receives an update from other BGP speakers in its own AS (iBGP), the BGP speaker that receives the update does not redistribute that information to other BGP speakers in its own AS. The BGP speaker that receives the update redistributes the information to other BGP speakers outside of its AS. Therefore, sustain a full mesh between the iBGP speakers within an AS.

In the diagram in this section, RTA and RTB run iBGP. RTA and RTD also run iBGP. The BGP updates that come from RTB to RTA transmit to RTE, which is outside the AS. The updates do not

transmit to RTD, which is inside the AS. Therefore, make an iBGP peering between RTB and RTD in order to not break the flow of the updates.

The BGP Decision Algorithm

After BGP receives updates about different destinations from different autonomous systems, the protocol must choose paths to reach a specific destination. BGP chooses only a single path to reach a specific destination.

BGP bases the decision on different **attributes**, such as next hop, administrative weights, local preference, route origin, path length, origin code, metric, and other attributes.

BGP always propagates the best path to the neighbors. Refer to BGP Best Path Selection Algorithm for more information.

The section BGP Case Studies 2 explains these attributes and their use.