

Contents

[Introduction](#)

[Prerequisites](#)

[Requirements](#)

[Components Used](#)

[Conventions](#)

[BGP Case Studies 1](#)

[How Does BGP Work?](#)

[eBGP and iBGP](#)

[Enable BGP Routing](#)

[Form BGP Neighbors](#)

[BGP and Loopback Interfaces](#)

[eBGP Multihop](#)

[eBGP Multihop \(Load Balancing\)](#)

[Route Maps](#)

[match and set Configuration Commands](#)

[network Command](#)

[Redistribution](#)

[Static Routes and Redistribution](#)

[iBGP](#)

[The BGP Decision Algorithm](#)

[BGP Case Studies 2](#)

[AS_PATH Attribute](#)

[Origin Attribute](#)

[BGP Next Hop Attribute](#)

[BGP Backdoor](#)

[Synchronization](#)

[Weight Attribute](#)

[Local Preference Attribute](#)

[Metric Attribute](#)

[Community Attribute](#)

[BGP Case Studies 3](#)

[BGP Filtering](#)

[AS Regular Expression](#)

[BGP Neighbors and Route Maps](#)

[BGP Case Studies 4](#)

[CIDR and Aggregate Addresses](#)

[BGP Confederation](#)

[Route Reflectors](#)

[Route Flap Dampening](#)

[How BGP Selects a Path](#)

[BGP Case Studies 5](#)

[Practical Design Example](#)

[Related Information](#)

[Related Cisco Support Community Discussions](#)

Introduction

This document contains five Border Gateway Protocol (BGP) case studies.

Prerequisites

Requirements

There are no specific requirements for this document.

Components Used

This document is not restricted to specific software and hardware versions.

Conventions

Refer to [Cisco Technical Tips Conventions](#) for more information on document conventions.

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

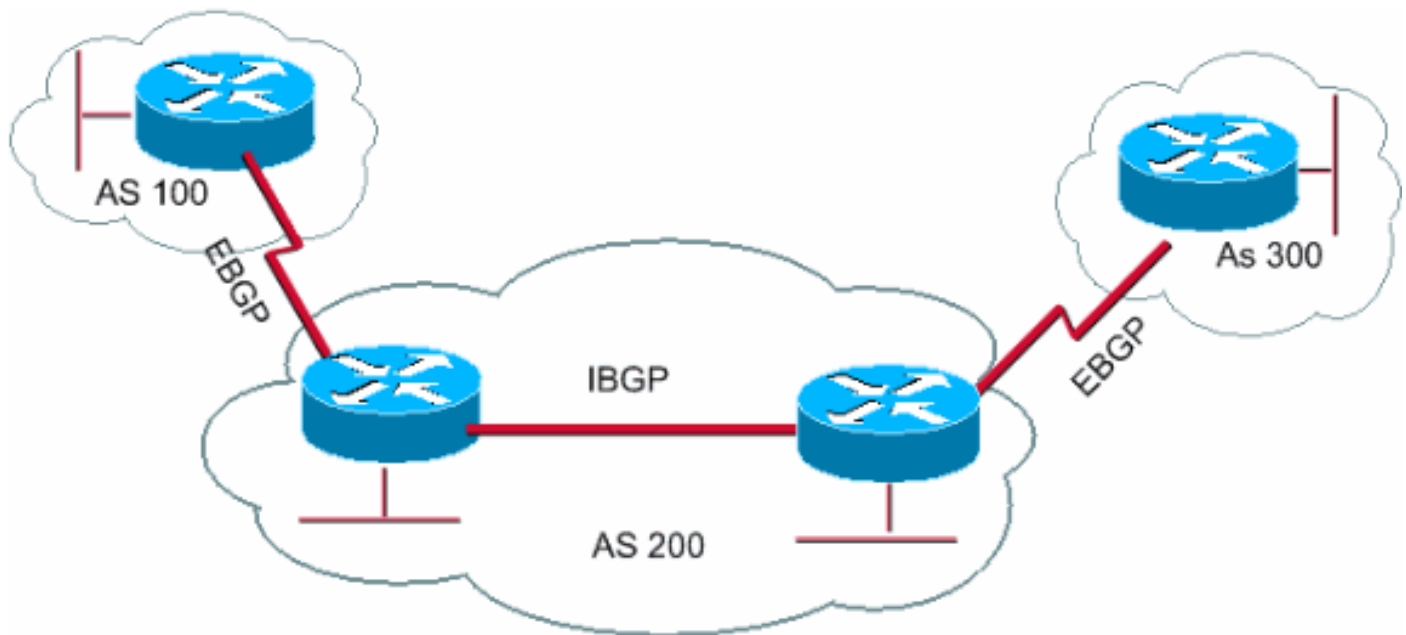
diagram in this section shows, AS200 is a transit AS for AS100 and AS300.

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.

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.

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.

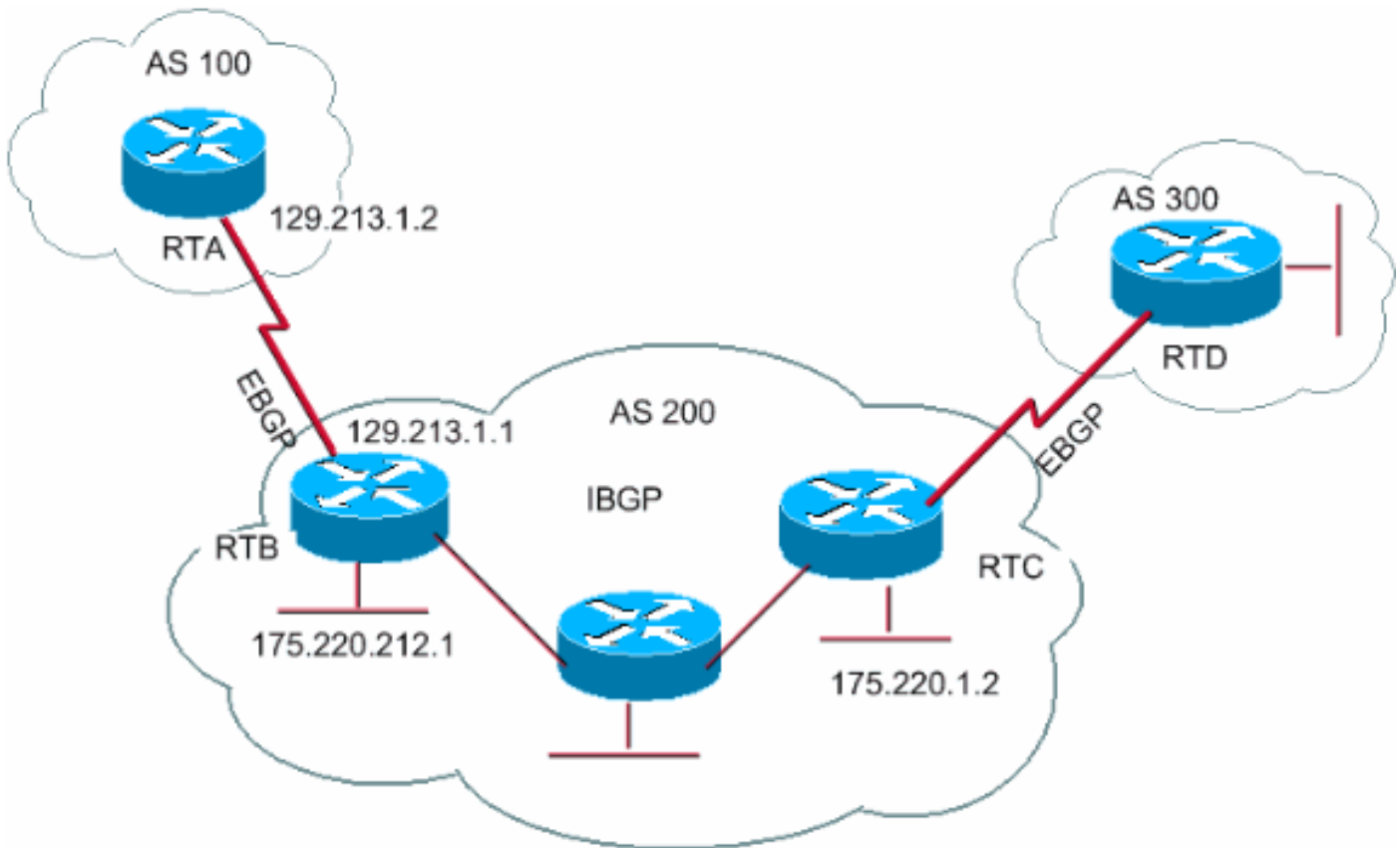
```
clear ip bgp *
```

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
```

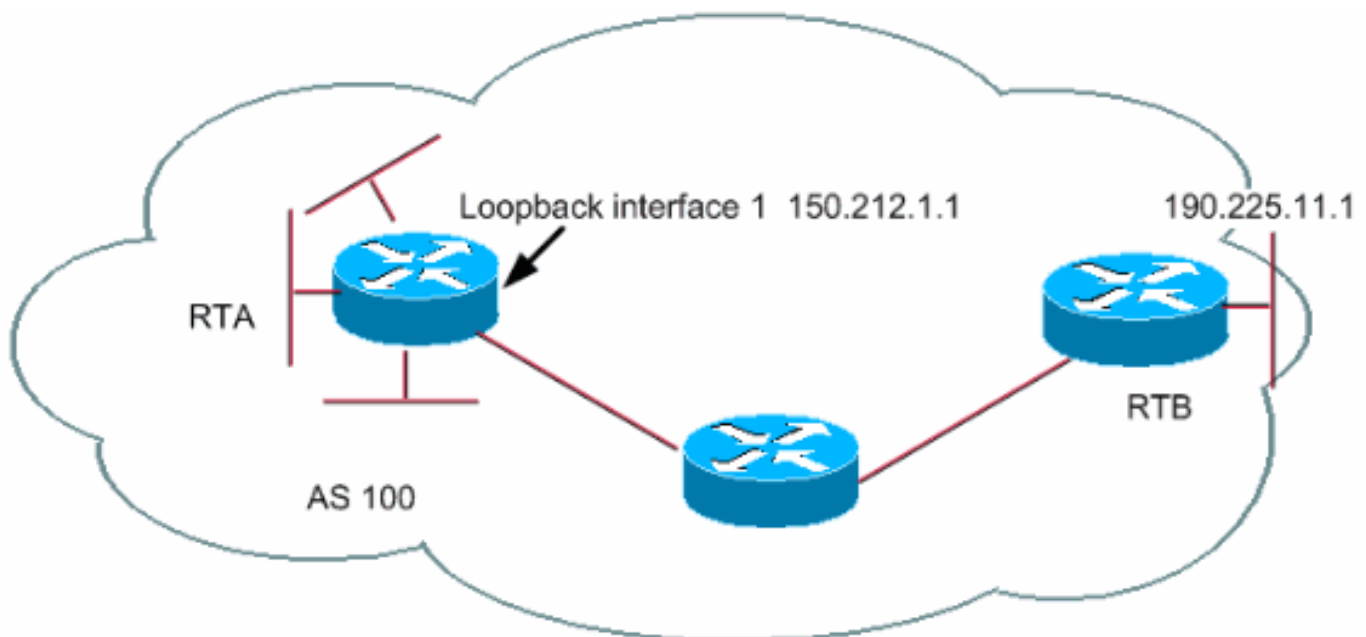
BGP and Loopback Interfaces

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-type interface-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

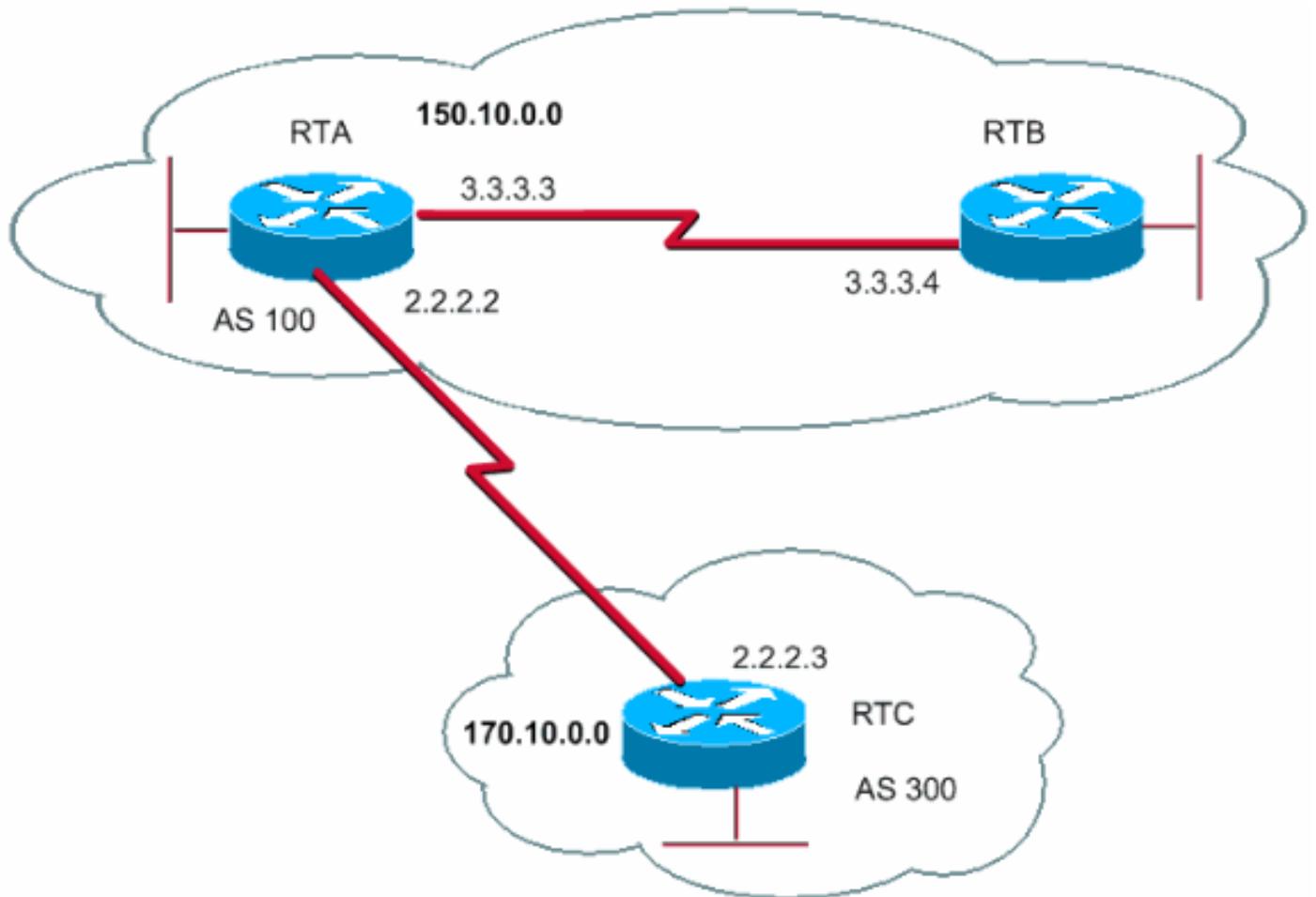
set next-hop

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
```

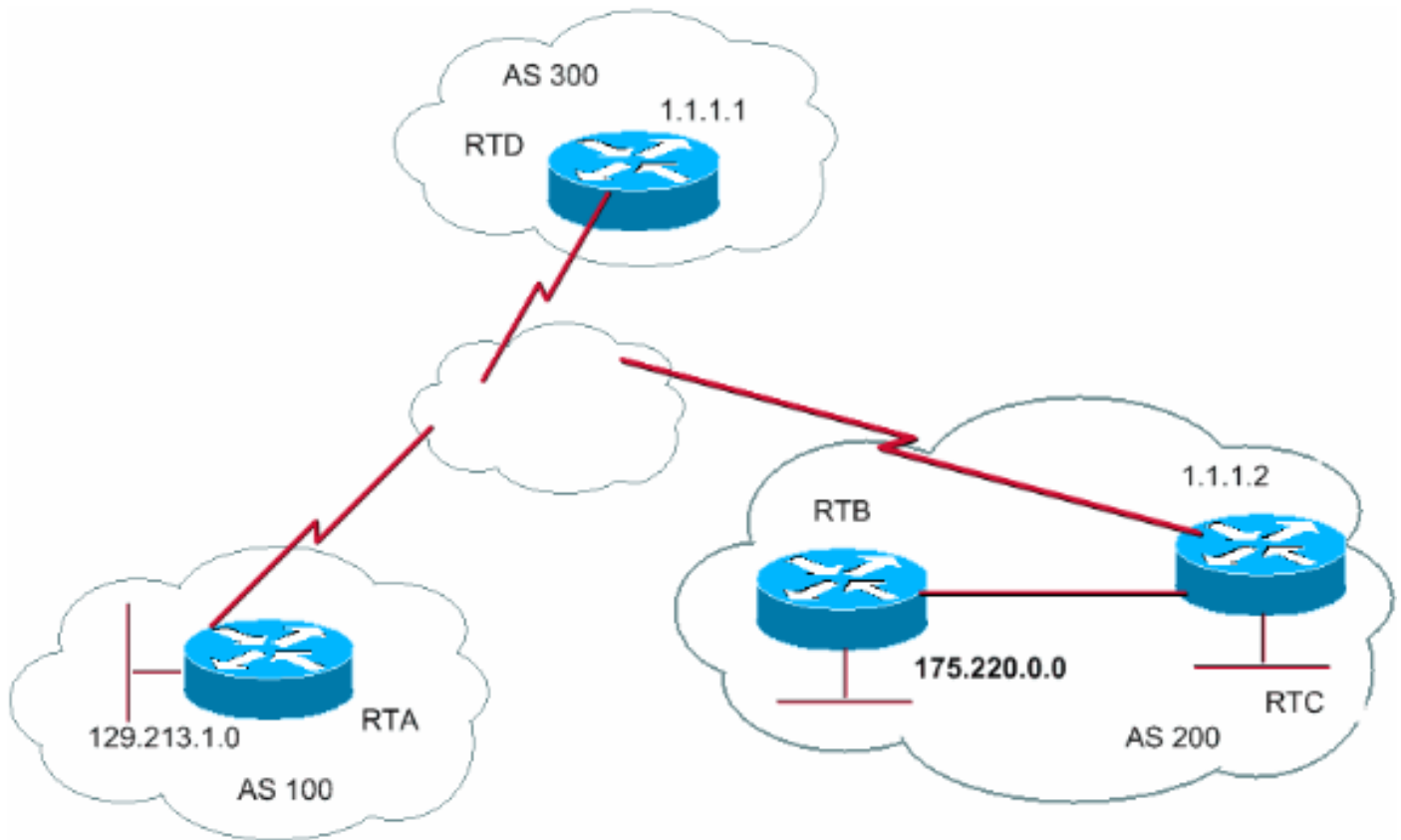
This example indicates that router A generates a network entry for 192.213.0.0/16. The /16 indicates that you use a supernet of the class C address and you advertise the first two octets, or first 16 bits.

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

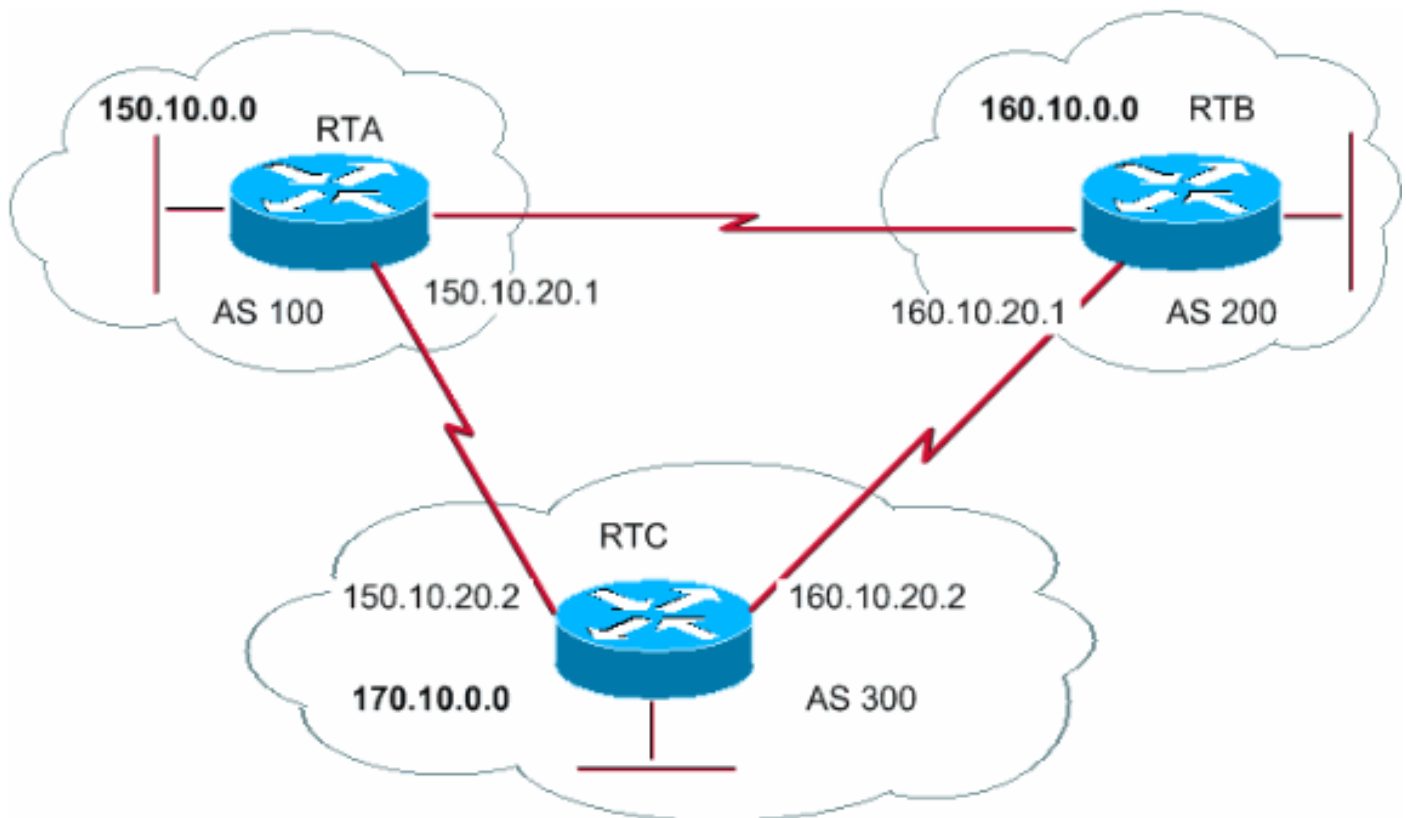
router bgp 200
neighbor 1.1.1.1 remote-as 300
redistribute static
...
ip route 175.220.0.0 255.255.255.0 null0
....
```

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 170.10.0.0

```

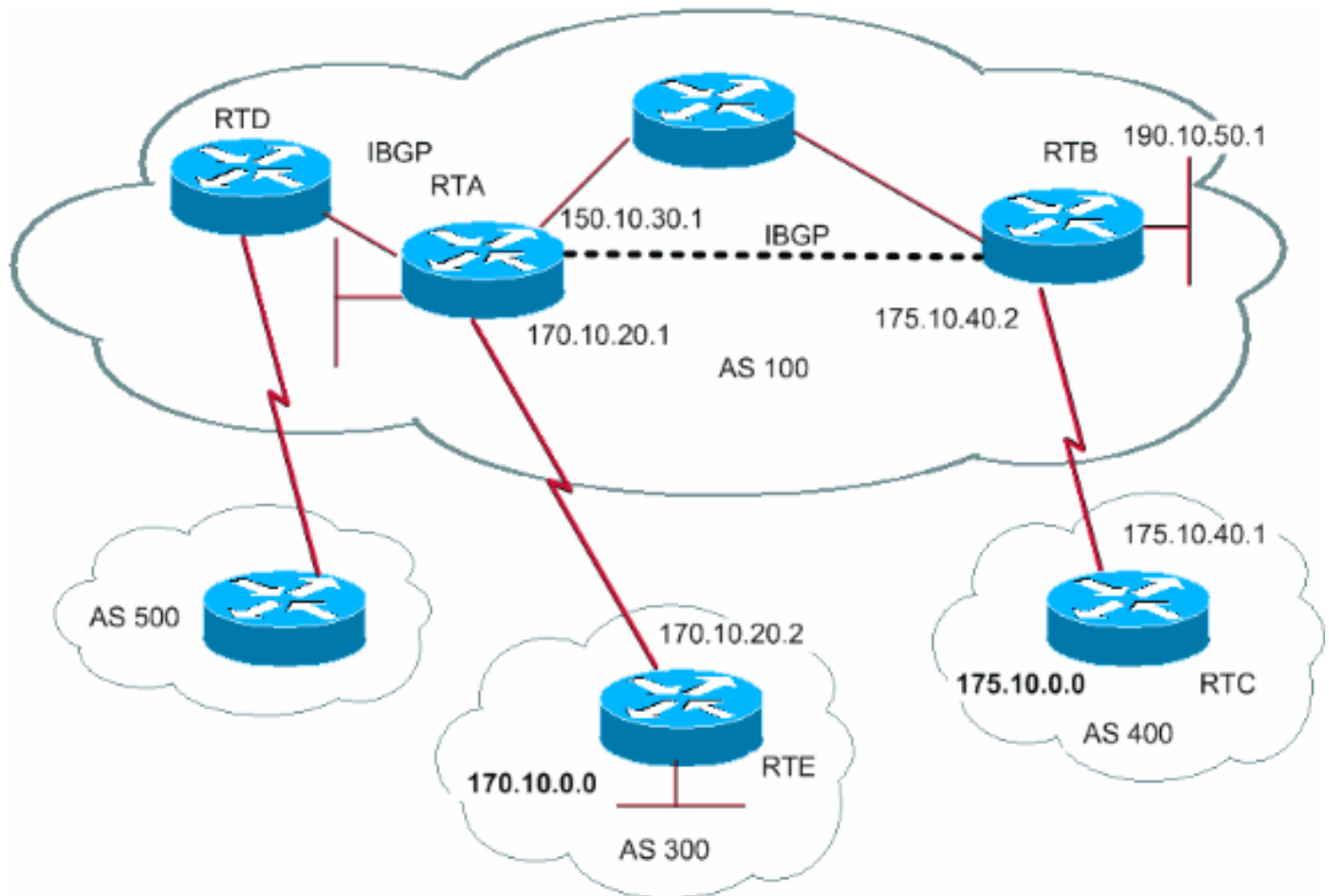
Note: You do not need network 150.10.0.0 or network 160.10.0.0 in RTC unless you want RTC to generate these networks as well as pass on these networks as they come in from AS100 and AS200. Again, the difference is that the **network** command adds an extra advertisement for these same networks, which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

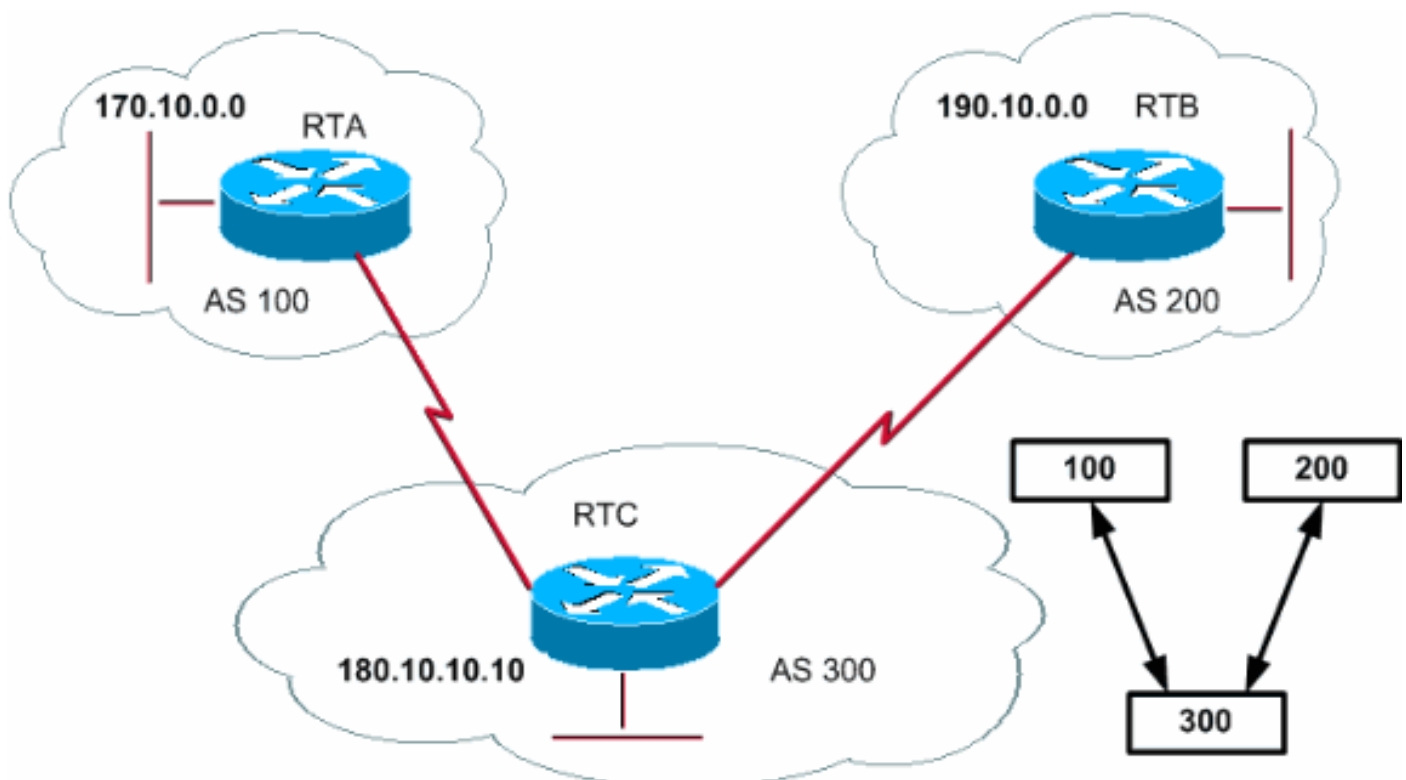
BGP bases the decision on different **attributes**, such as next hop, administrative weights, local preference, route origin, path length, origin code, metric, and other attributes.

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.

BGP Case Studies 2

AS PATH Attribute



Whenever a route update passes through an AS, the AS number is prepended to that update. The AS_PATH attribute is actually the list of AS numbers that a route has traversed in order to reach a destination. An AS_SET is an ordered mathematical set $\{ \}$ of all the ASs that have been traversed. The [CIDR Example 2 \(as-set\)](#) section of this document provides an example of AS_SET.

In the example in this section, RTB advertises network 190.10.0.0 in AS200. When that route

traverses AS300, RTC appends its own AS number to the network. So when 190.10.0.0 reaches RTA, the network has two AS numbers attached: first 200, then 300. For RTA, the path to reach 190.10.0.0 is (300, 200).

The same process applies to 170.10.0.0 and 180.10.0.0. RTB has to take path (300, 100); RTB traverses AS300 and then AS100 in order to reach 170.10.0.0. RTC has to traverse path (200) in order to reach 190.10.0.0 and path (100) in order to reach 170.10.0.0.

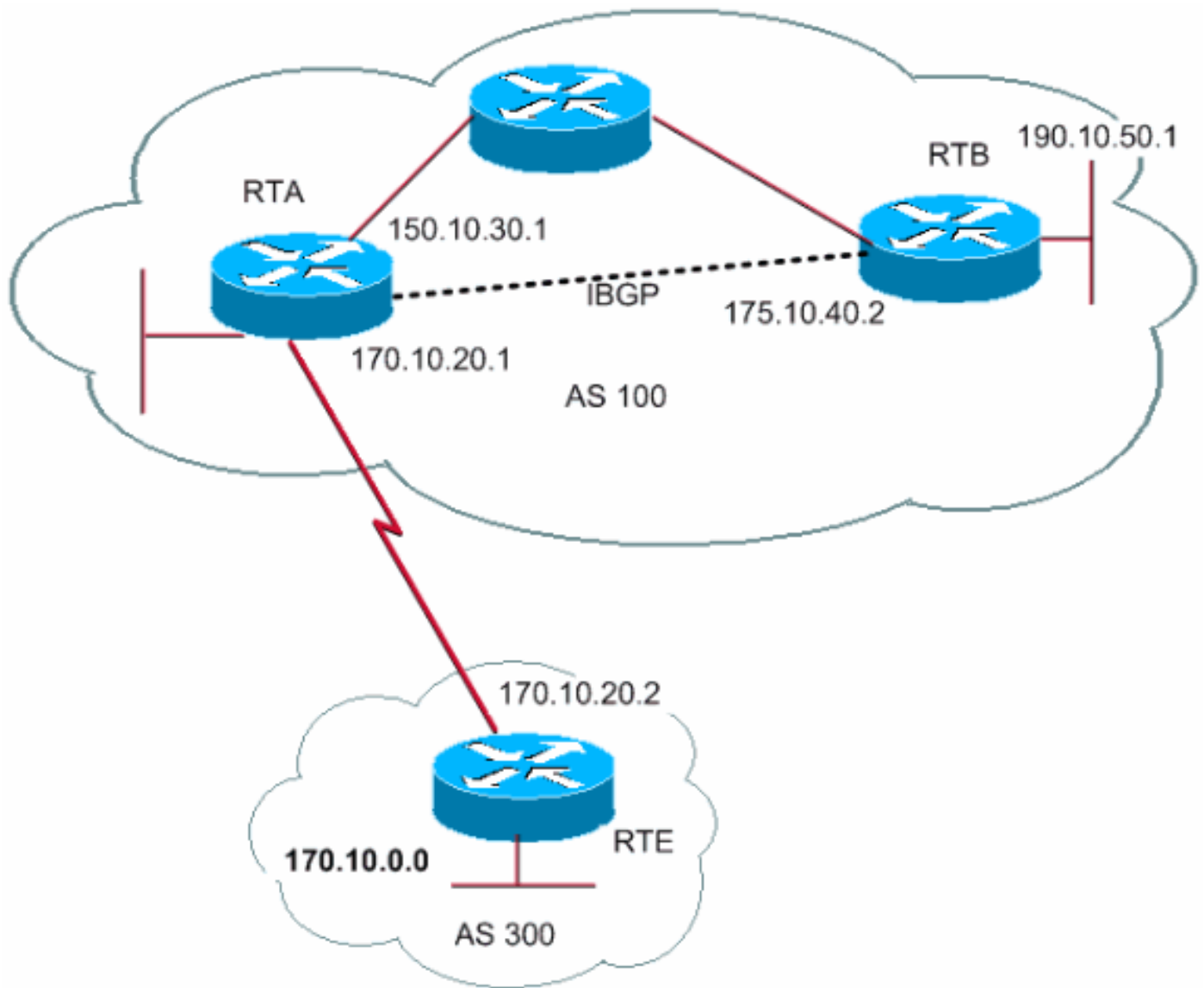
Origin Attribute

The origin is a mandatory attribute that defines the origin of the path information. The origin attribute can assume three values:

IGP—Network Layer Reachability Information (NLRI) is interior to the AS of origination. This normally happens when you issue the **bgp network** command . An **i** in the BGP table indicates IGP.

EGP—NLRI is learned via exterior gateway protocol (EGP). An **e** in the BGP table indicates EGP.

INCOMPLETE—NLRI is unknown or learned via some other means. INCOMPLETE usually occurs when you redistribute routes from other routing protocols into BGP and the origin of the route is incomplete. An **?** in the BGP table indicates INCOMPLETE.



```

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
redistribute static

ip route 190.10.0.0 255.255.0.0 null0

```

```

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

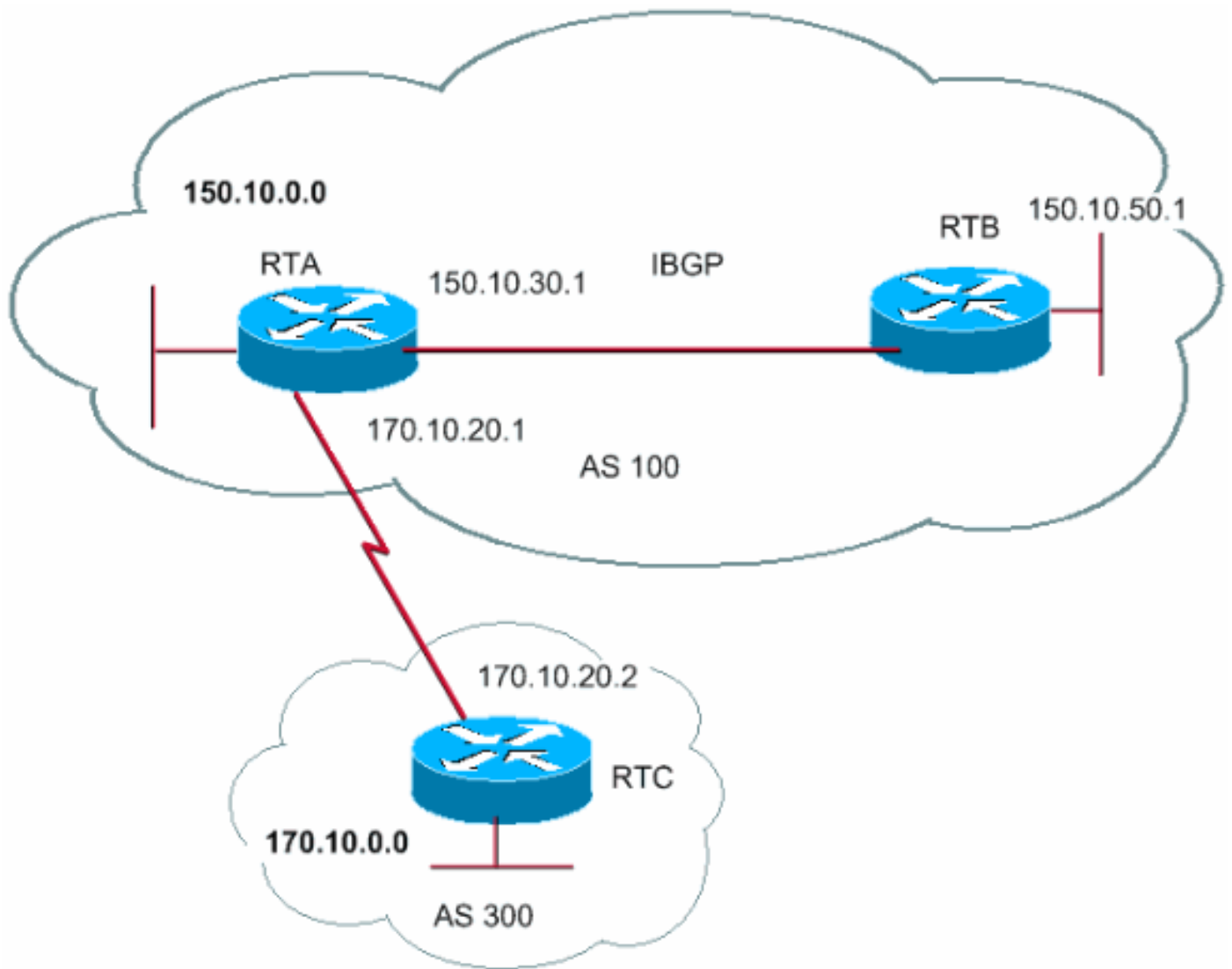
RTE#
router bgp 300
neighbor 170.10.20.1 remote-as 100
network 170.10.0.0

```

RTA reaches 170.10.0.0 via 300 i. The "300 i" means that the next AS path is 300 and the origin of the route is IGP. RTA also reaches 190.10.50.0 via i. This "i" means that the entry is in the same AS and the origin is IGP. RTE reaches 150.10.0.0 via 100 i. The "100 i" means that the next AS is 100 and the origin is IGP. RTE also reaches 190.10.0.0 via 100 ?. The "100 ?" means that the

next AS is 100 and that the origin is incomplete and comes from a static route.

BGP Next Hop Attribute



The BGP next hop attribute is the next hop IP address to use in order to reach a certain destination.

For eBGP, the next hop is always the IP address of the neighbor that the **neighbor** command specifies. In the example in this section, RTC advertises 170.10.0.0 to RTA with a next hop of 170.10.20.2. RTA advertises 150.10.0.0 to RTC with a next hop of 170.10.20.1. For iBGP, the protocol states that the next hop that eBGP advertises should be carried into iBGP. Because of this rule, RTA advertises 170.10.0.0 to its iBGP peer RTB with a next hop of 170.10.20.2. So, according to RTB, the next hop to reach 170.10.0.0 is 170.10.20.2 and *not* 150.10.30.1.

Make sure that RTB can reach 170.10.20.2 via IGP. Otherwise, RTB drops packets with the destination of 170.10.0.0 because the next hop address is inaccessible. For example, if RTB runs iGRP, you can also run iGRP on RTA network 170.10.0.0. You want to make iGRP passive on the link to RTC so that BGP is only exchanged.

```
RTA#  
router bgp 100  
neighbor 170.10.20.2 remote-as 300  
neighbor 150.10.50.1 remote-as 100
```

```

network 150.10.0.0
RTB#
router bgp 100
neighbor 150.10.30.1 remote-as 100
RTC#
router bgp 300
neighbor 170.10.20.1 remote-as 100
network 170.10.0.0

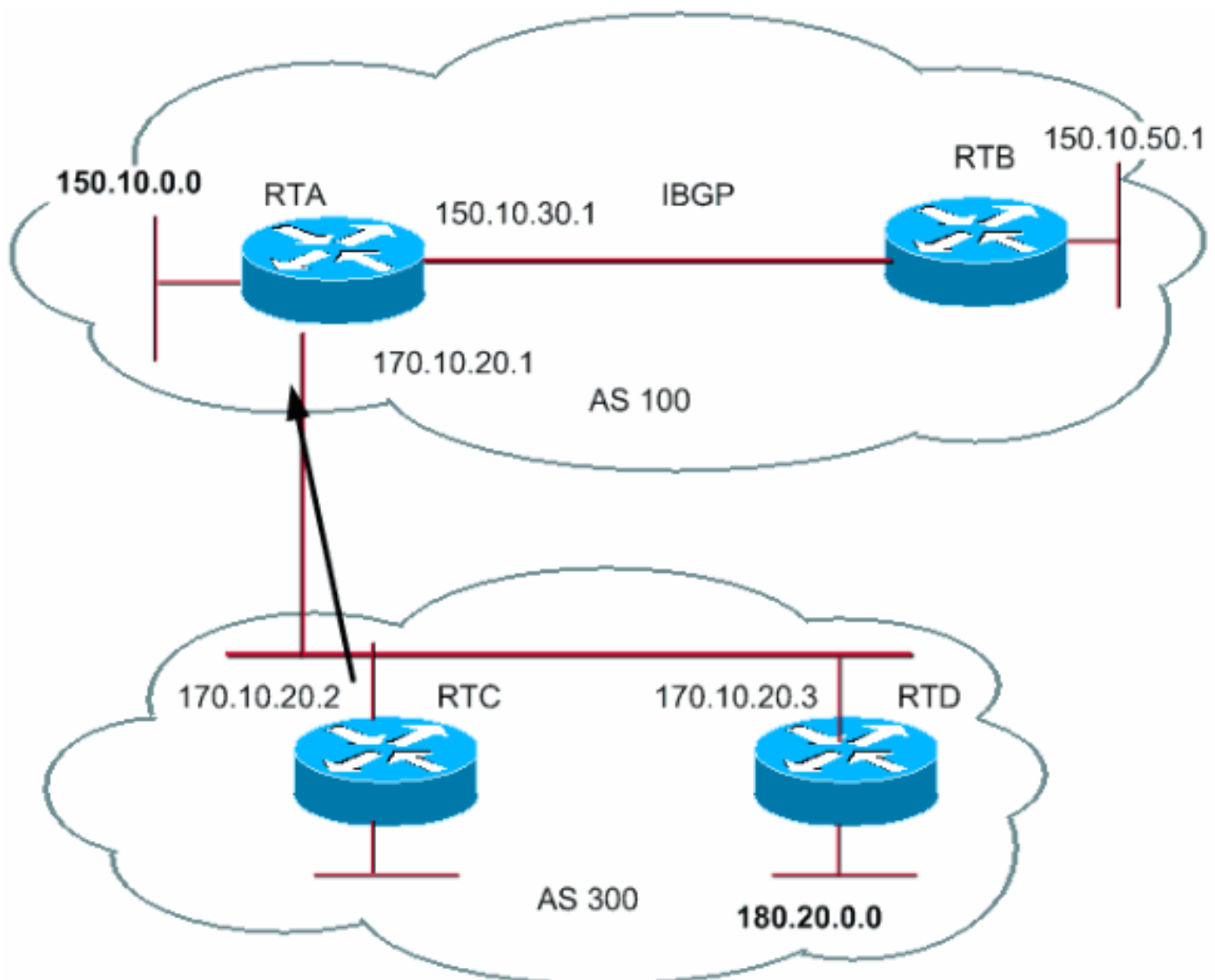
```

Note: RTC advertises 170.10.0.0 to RTA with a next hop equal to 170.10.20.2.

Note: RTA advertises 170.10.0.0 to RTB with a next hop equal to 170.10.20.2. The eBGP next hop is carried in iBGP.

Take special care when you deal with multiaccess and nonbroadcast multiaccess (NBMA) networks. The sections [BGP Next Hop \(Multiaccess Networks\)](#) and [BGP Next Hop \(NBMA\)](#) provide more details.

[BGP Next Hop \(Multiaccess Networks\)](#)



This example shows how the next hop behaves on a multiaccess network such as Ethernet.

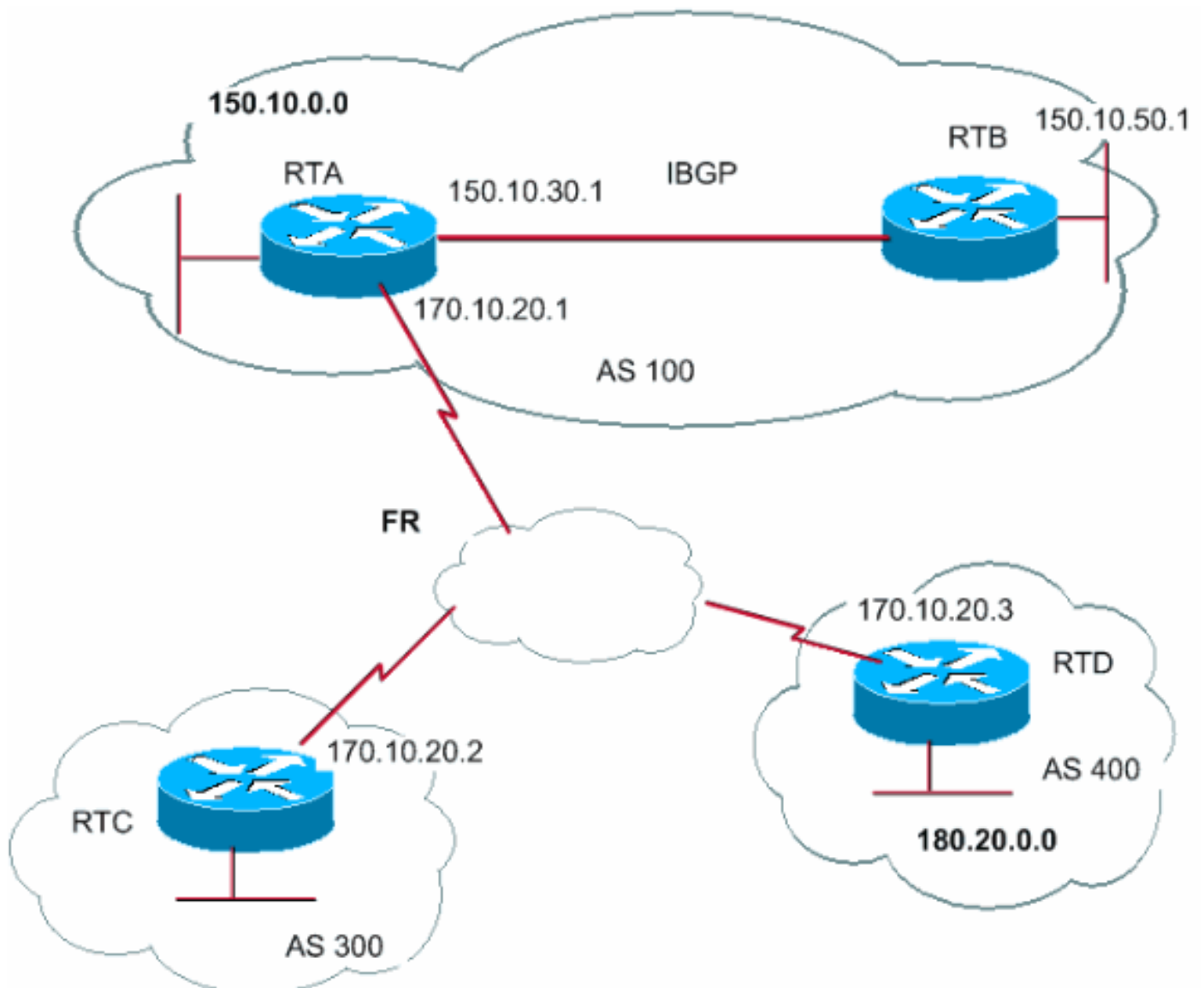
Assume that RTC and RTD in AS300 run OSPF. RTC runs BGP with RTA. RTC can reach

network 180.20.0.0 via 170.10.20.3. When RTC sends a BGP update to RTA with regard to 180.20.0.0, RTC uses as next hop 170.10.20.3. RTC does not use its own IP address, 170.10.20.2. RTC uses this address because the network between RTA, RTC, and RTD is a multiaccess network. The RTA use of RTD as a next hop to reach 180.20.0.0 is more sensible than the extra hop via RTC.

Note: RTC advertises 180.20.0.0 to RTA with a next hop 170.10.20.3.

If the common medium to RTA, RTC, and RTD is not multiaccess, but NBMA, further complications occur.

BGP Next Hop (NBMA)



The common medium appears as a cloud in the diagram. If the common medium is a frame relay or any NBMA cloud, the exact behavior is as if you have connection via Ethernet. RTC advertises 180.20.0.0 to RTA with a next hop of 170.10.20.3.

The problem is that RTA does not have a direct permanent virtual circuit (PVC) to RTD and cannot reach the next hop. In this case, routing fails.

The **next-hop-self** command remedies this situation.

next-hop-self Command

For situations with the next hop, as in the [BGP Next Hop \(NBMA\)](#) example, you can use the **next-hop-self** command. The syntax is:

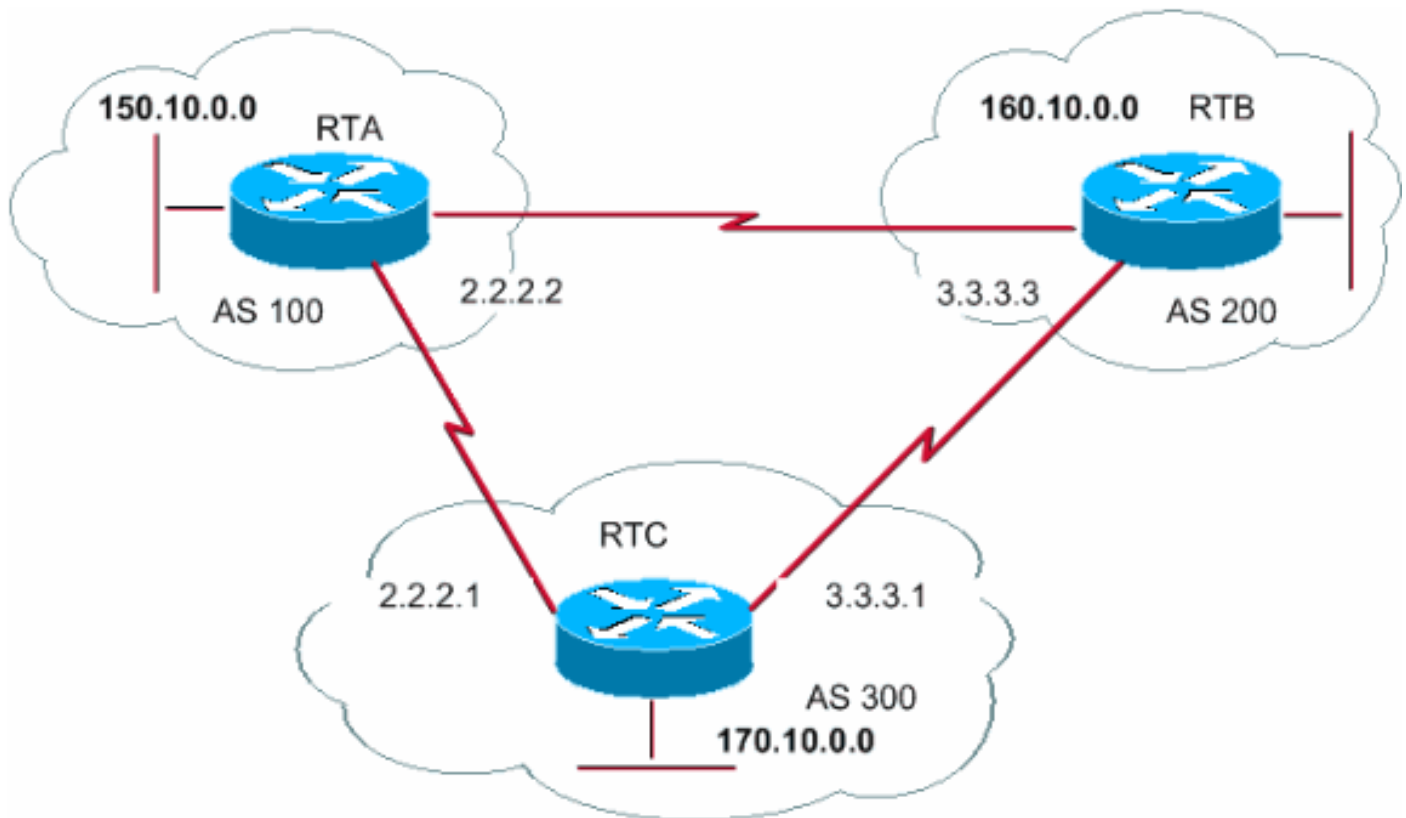
```
neighbor {ip-address | peer-group-name} next-hop-self
```

The **next-hop-self** command allows you to force BGP to use a specific IP address as the next hop.

For the [BGP Next Hop \(NBMA\)](#) example, this configuration solves the problem:

```
RTC#  
router bgp 300  
neighbor 170.10.20.1 remote-as 100  
neighbor 170.10.20.1 next-hop-self  
RTC advertises 180.20.0.0 with a next hop equal to 170.10.20.2.
```

BGP Backdoor



In this diagram, RTA and RTC run eBGP. RTB and RTC run eBGP. RTA and RTB run some kind of IGP, either RIP, IGRP, or another protocol. By definition, eBGP updates have a distance of 20, which is less than the IGP distances. The default distances are:

120 for RIP

100 for IGRP

90 for EIGRP

110 for OSPF

RTA receives updates about 160.10.0.0 via two routing protocols:

eBGP with a distance of 20

IGP with a distance that is greater than 20

By default, BGP has these distances:

External distance—20

Internal distance—200

Local distance—200

But you can use the **distance** command to change the default distances:

```
distance bgp external-distance internal-distance local-distance
```

RTA picks eBGP via RTC because of the shorter distance.

If you want RTA to learn about 160.10.0.0 via RTB (IGP), then you have two options:

Change the external distance of eBGP or the IGP distance.

Note: This change is not recommended.

Use BGP backdoor.

BGP backdoor makes the IGP route the preferred route.

Issue the [network address backdoor](#) command.

The configured network is the network that you want to reach via IGP. For BGP, this network gets the same treatment as a locally assigned network, except BGP updates do not advertise this network.

```
RTA#  
router eigrp 10  
  
network 150.10.0.0
```

```

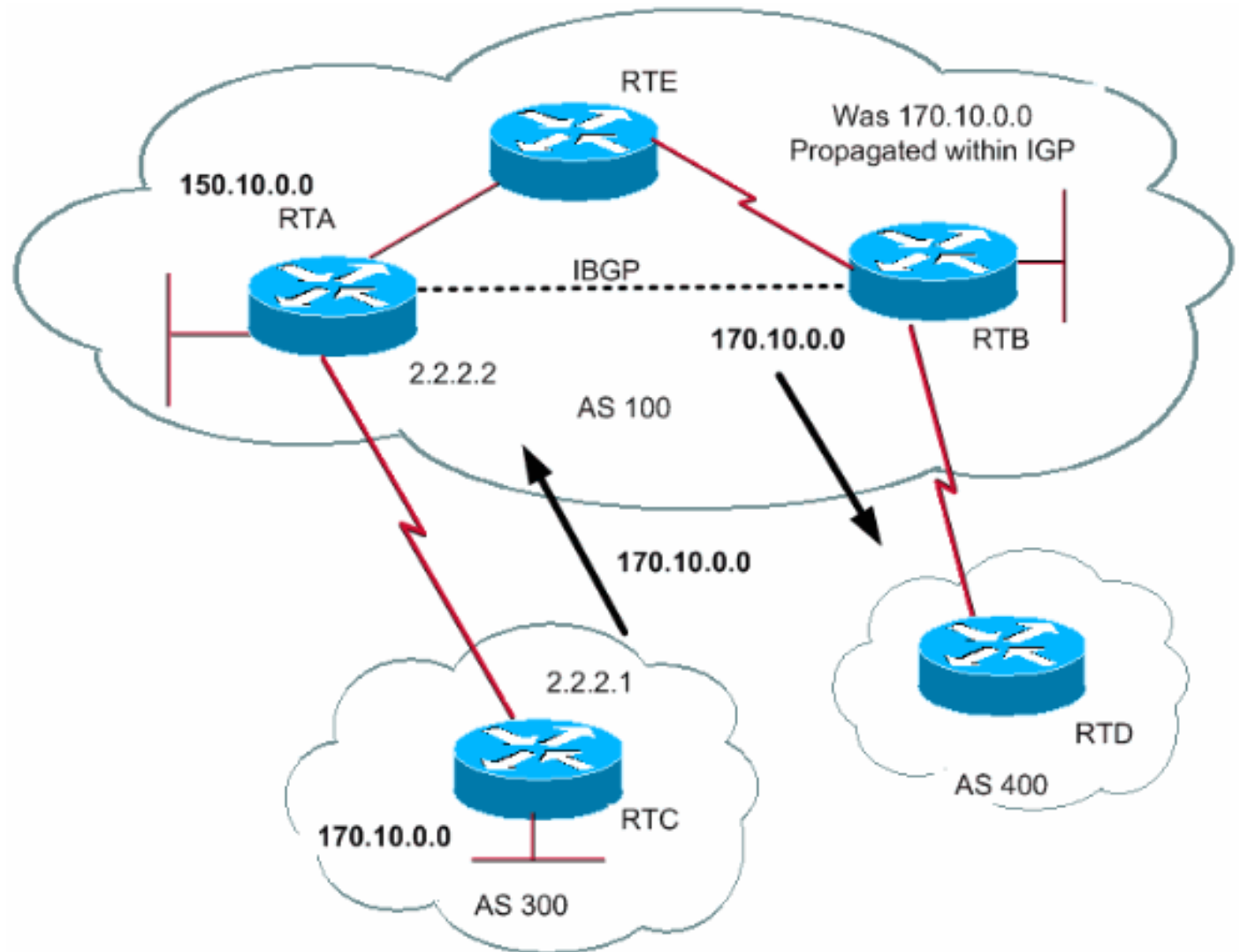
router bgp 100
neighbor 2.2.2.1 remote-as 300
network 160.10.0.0 backdoor

```

Network 160.10.0.0 is treated as a local entry, but is not advertised as a normal network entry.

RTA learns 160.10.0.0 from RTB via EIGRP with distance 90. RTA also learns the address from RTC via eBGP with distance 20. Normally eBGP is the preference, but because of the **network backdoor** command, EIGRP is the preference.

Synchronization



Before the discussion of synchronization, look at this scenario. RTC in AS300 sends updates about 170.10.0.0. RTA and RTB run iBGP, so RTB gets the update and is able to reach 170.10.0.0 via next hop 2.2.2.1. Remember that the next hop is carried via iBGP. In order to reach the next hop, RTB must send the traffic to RTE.

Assume that RTA has not redistributed network 170.10.0.0 into IGP. At this point, RTE has no idea that 170.10.0.0 even exists.

If RTB starts to advertise to AS400 that RTB can reach 170.10.0.0, traffic that comes from RTD to RTB with destination 170.10.0.0 flows in and drops at RTE.

Synchronization states that, if your AS passes traffic from another AS to a third AS, BGP should not advertise a route before all the routers in your AS have learned about the route via IGP. BGP

waits until IGP has propagated the route within the AS. Then, BGP advertises the route to external peers.

In the example in this section, RTB waits to hear about 170.10.0.0 via IGP. Then, RTB starts to send the update to RTD. You can make RTB think that IGP has propagated the information if you add a static route in RTB that points to 170.10.0.0. Make sure that other routers can reach 170.10.0.0.

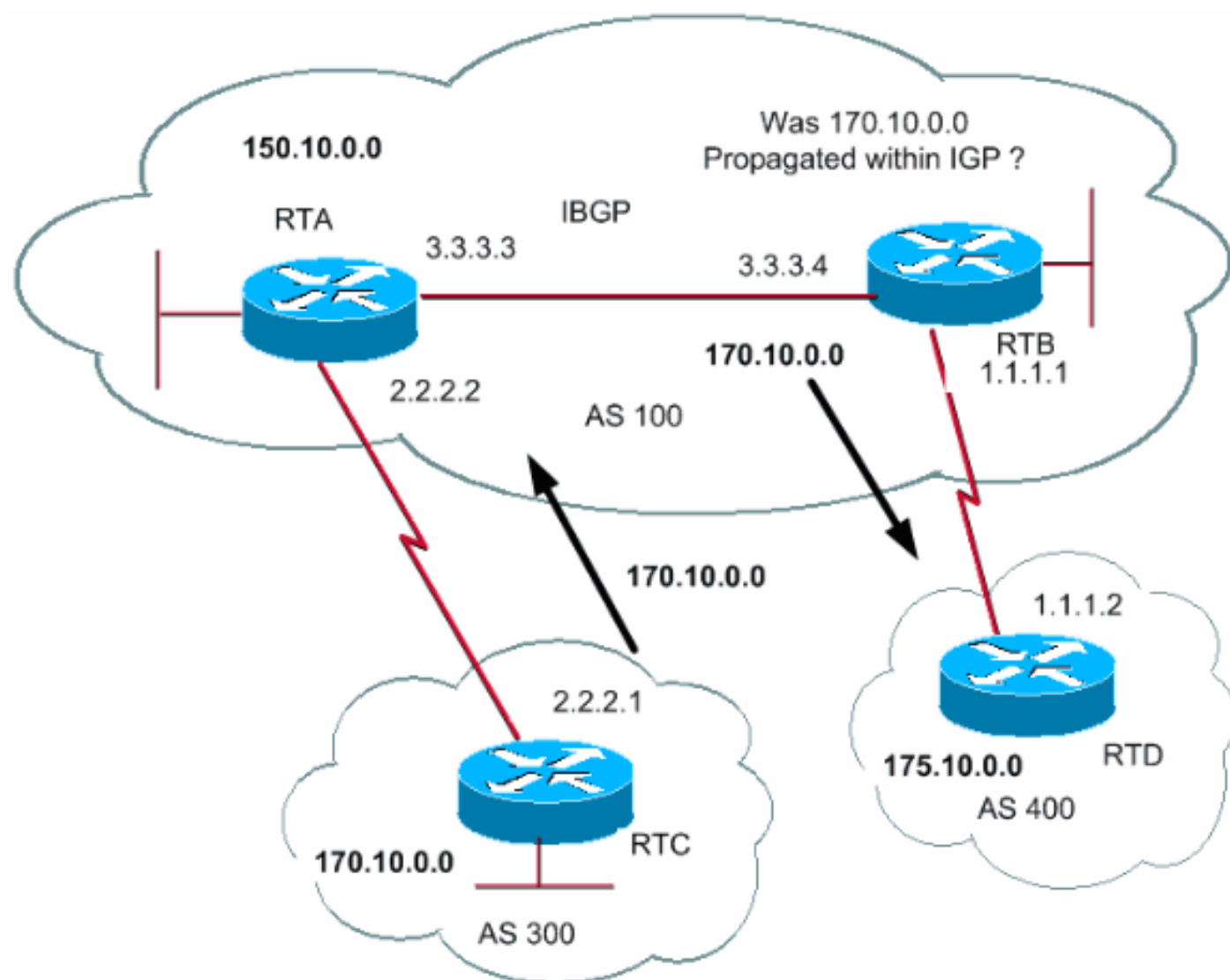
Disable Synchronization

In some cases, you do not need synchronization. If you do not pass traffic from a different AS through your AS, you can disable synchronization. You can also disable synchronization if all routers in your AS run BGP. The disablement of this feature can allow you to carry fewer routes in your IGP and allow BGP to converge more quickly.

The disablement of synchronization is not automatic. If all your routers in the AS run BGP and you do not run IGP at all, the router has no way to know. Your router waits indefinitely for an IGP update about a certain route before the router sends the route to external peers. You have to disable synchronization manually in this case so that routing can work correctly:

```
router bgp 100
no synchronization
```

Note: Make sure that you issue the **clear ip bgp address** command to reset the session.

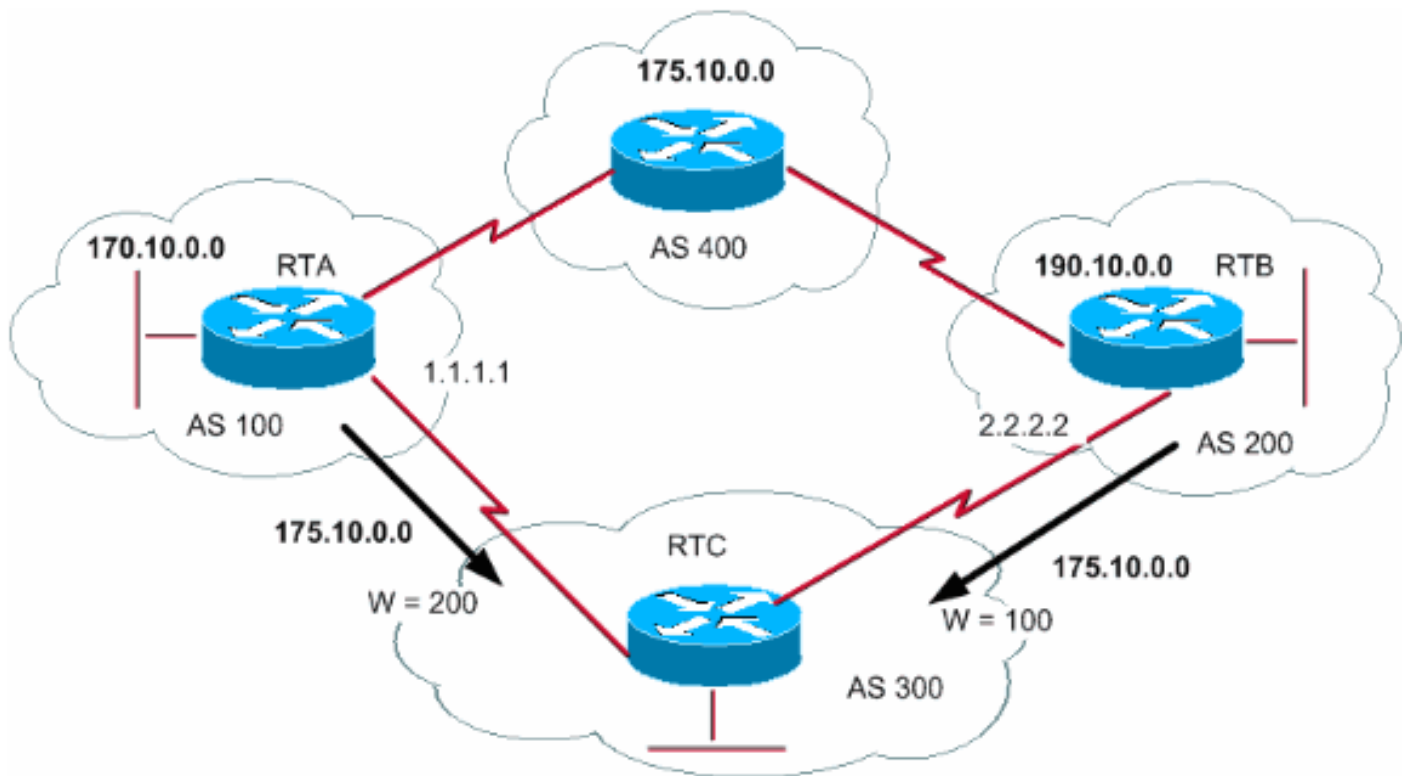


```

RTB#
router bgp 100
network 150.10.0.0
neighbor 1.1.1.2 remote-as 400
neighbor 3.3.3.3 remote-as 100
no synchronization
!--- RTB puts 170.10.0.0 in its IP routing table and advertises the network !--- to RTD, even if
RTB does not have an IGP path to 170.10.0.0.
RTD# router bgp 400 neighbor 1.1.1.1 remote-as 100
network 175.10.0.0
RTA# router bgp 100 network 150.10.0.0 neighbor 3.3.3.4 remote-as 100

```

Weight Attribute



The weight attribute is a Cisco-defined attribute. This attribute uses weight to select a best path. The weight is assigned locally to the router. The value only makes sense to the specific router. The value is not propagated or carried through any of the route updates. A weight can be a number from 0 to 65,535. Paths that the router originates have a weight of 32,768 by default, and other paths have a weight of 0.

Routes with a higher weight value have preference when multiple routes to the same destination exist. Look at the example in this section. RTA has learned about network 175.10.0.0 from AS4. RTA propagates the update to RTC. RTB has also learned about network 175.10.0.0 from AS4. RTB propagates the update to RTC. RTC now has two ways to reach 175.10.0.0 and has to decide which way to go. If you set the weight of the updates on RTC that come from RTA so that the weight is greater than the weight of updates that come from RTB, you force RTC to use RTA as a next hop to reach 175.10.0.0. Multiple methods achieve this weight set:

Use the **neighbor** command.

```
neighbor {ip-address | peer-group} weight weight
```

Use AS_PATH access lists.

```
ip as-path access-list access-list-number {permit | deny} as-regular-expression  
neighbor ip-address filter-list access-list-number weight weight
```

Use route maps.

```
RTC#  
router bgp 300  
neighbor 1.1.1.1 remote-as 100  
neighbor 1.1.1.1 weight 200  
!--- The route to 175.10.0.0 from RTA has a 200 weight. neighbor 2.2.2.2 remote-as 200  
neighbor 2.2.2.2 weight 100 !--- The route to 175.10.0.0 from RTB has a 100 weight.
```

RTA, which has a higher weight value, has preference as the next hop.


You can achieve the same outcome with IP AS_PATH and filter lists.

```
RTC#  
router bgp 300  
neighbor 1.1.1.1 remote-as 100  
neighbor 1.1.1.1 filter-list 5 weight 200  
neighbor 2.2.2.2 remote-as 200  
neighbor 2.2.2.2 filter-list 6 weight 100  
...  
ip as-path access-list 5 permit ^100$  
!--- This only permits path 100. ip as-path access-list 6 permit ^200$ ...
```

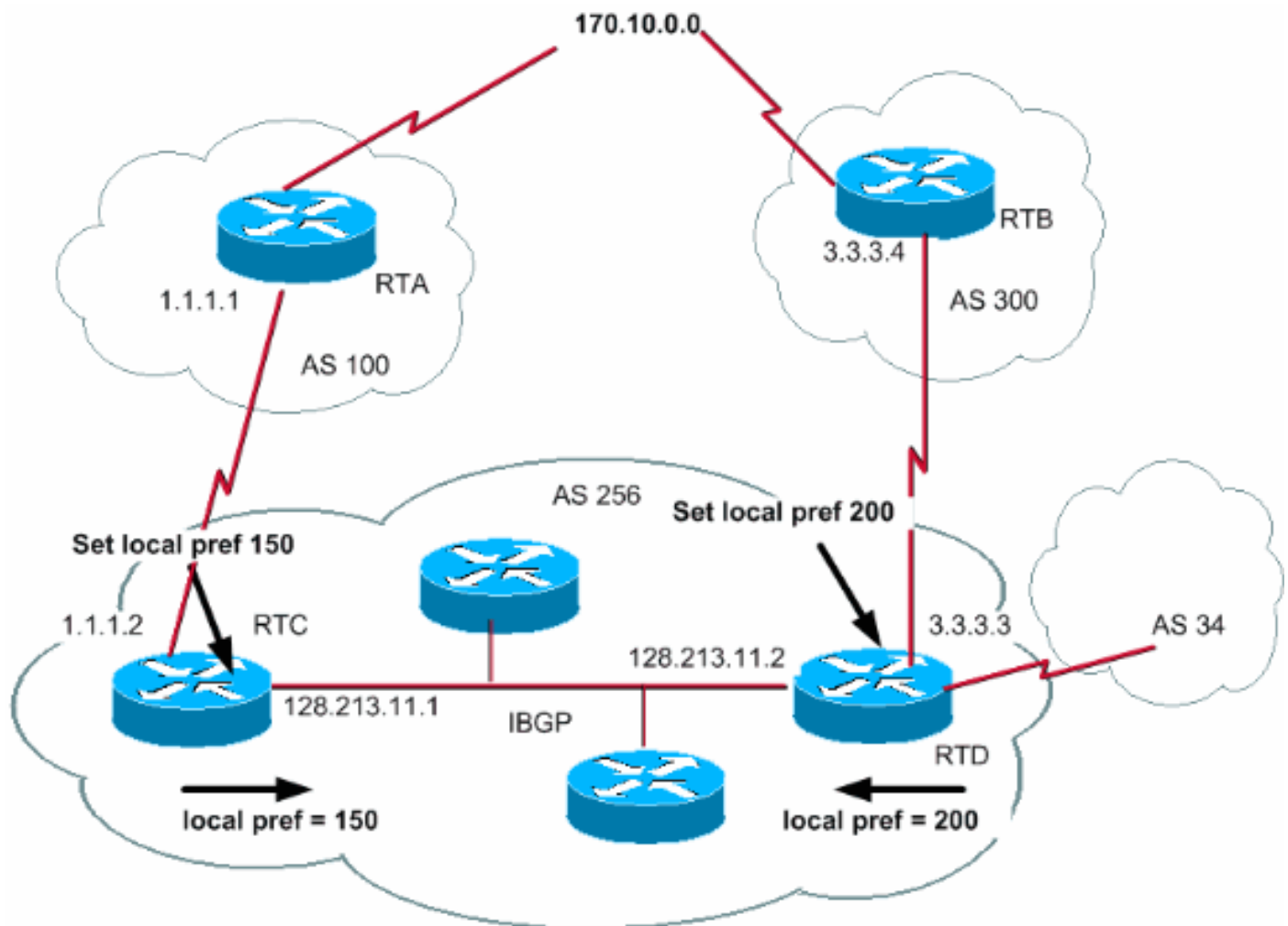
You also can achieve the same outcome with the use of route maps.

```
RTC#  
router bgp 300  
neighbor 1.1.1.1 remote-as 100  
neighbor 1.1.1.1 route-map setweightin in  
neighbor 2.2.2.2 remote-as 200  
neighbor 2.2.2.2 route-map setweightin in  
...  
ip as-path access-list 5 permit ^100$  
...  
  
route-map setweightin permit 10  
match as-path 5  
set weight 200  
!--- Anything that applies to access list 5, such as packets from AS100, has weight 200. route-  
map setweightin permit 20 set weight 100 !--- Anything else has weight 100.
```

Note: You can modify weight to prefer MPLS VPN BGP path with IGP path as a Backup.

Note: For more information, refer to this Cisco Support Community document that describes how to configure the router to have a preferred path on both primary and failure conditions and to reroute on primary path recovery: [Preferring MPLS VPN BGP Path with IGP Backup](#) 

[Local Preference Attribute](#)



Local preference is an indication to the AS about which path has preference to exit the AS in order to reach a certain network. A path with a higher local preference is preferred more. The default value for local preference is 100.

Unlike the weight attribute, which is only relevant to the local router, local preference is an attribute that routers exchange in the same AS.

You set local preference with the issue of the [bgp default local-preference](#) *value* command. You can also set local preference with route maps, as the example in this section demonstrates:

Note: It is necessary to perform a soft reset (that is, clear the bgp process on the router) in order for changes to be taken in to consideration. In order to clear the bgp process, use the [clear ip bgp \[soft\]\[in/out\]](#) command where **soft** indicates a soft reset without tearing the session and **[in/out]** specifies inbound or outbound configuration. If **in/out** is not specified both inbound and outbound sessions are reset.

The **bgp default local-preference** command sets the local preference on the updates out of the router that go to peers in the same AS. In the diagram in this section, AS256 receives updates about 170.10.0.0 from two different sides of the organization. Local preference helps you determine which way to exit AS256 in order to reach that network. Assume that RTD is the exit point preference. This configuration sets the local preference for updates that come from AS300 to 200 and for updates that come from AS100 to 150:

```
RTC#
router bgp 256
neighbor 1.1.1.1 remote-as 100
```

```
neighbor 128.213.11.2 remote-as 256
bgp default local-preference 150
```

```
RTD#
router bgp 256
neighbor 3.3.3.4 remote-as 300
neighbor 128.213.11.1 remote-as 256
bgp default local-preference 200
```

In this configuration, RTC sets the local preference of all updates to 150. The same RTD sets the local preference of all updates to 200. There is an exchange of local preference within AS256. Therefore, both RTC and RTD realize that network 170.10.0.0 has a higher local preference when updates come from AS300 rather than from AS100. All traffic in AS256 that has that network as a destination transmits with RTD as an exit point.

The use of route maps provides more flexibility. In the example in this section, all updates that RTD receives are tagged with local preference 200 when the updates reach RTD. Updates that come from AS34 also are tagged with the local preference of 200. This tag can be unnecessary. For this reason, you can use route maps to specify the specific updates that need to be tagged with a specific local preference. Here is an example:

```
RTD#
router bgp 256
neighbor 3.3.3.4 remote-as 300
neighbor 3.3.3.4 route-map setlocalin in
neighbor 128.213.11.1 remote-as 256
....
ip as-path access-list 7 permit ^300$
...

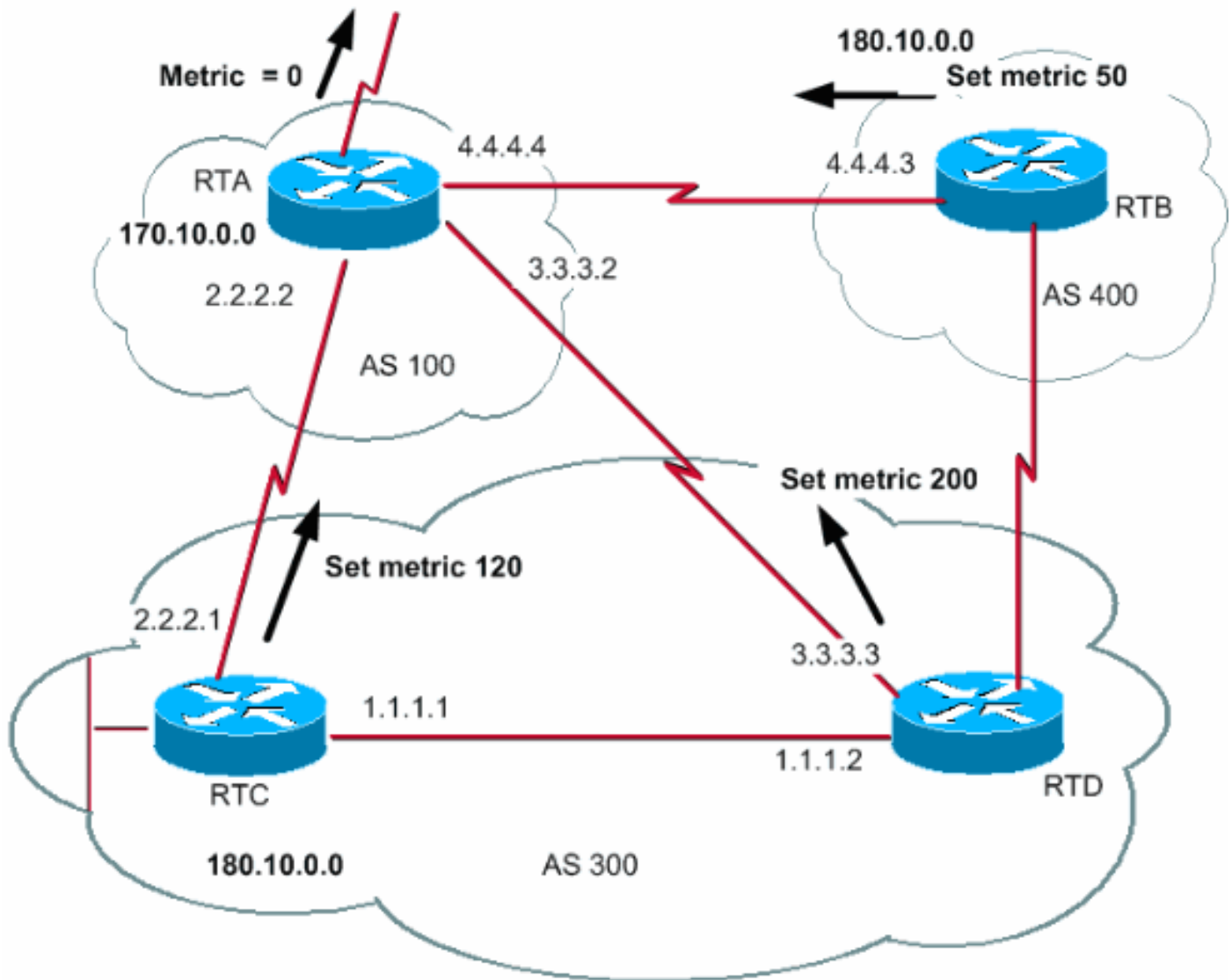
route-map setlocalin permit 10
match as-path 7
set local-preference 200

route-map setlocalin permit 20
set local-preference 150
```

With this configuration, any update that comes from AS300 has a local preference of 200. Any other updates, such as updates that come from AS34, have a value of 150.

[Metric Attribute](#)

METRIC (MULTI_EXIT_DISC) (INTER_AS)



The metric attribute also has the name MULTI_EXIT_DISCRIMINATOR, MED (BGP4), or INTER_AS (BGP3). The attribute is a hint to external neighbors about the path preference into an AS. The attribute provides a dynamic way to influence another AS in the way to reach a certain route when there are multiple entry points into that AS. A lower metric value is preferred more.

Unlike local preference, metric is exchanged between ASs. A metric is carried into an AS but does not leave the AS. When an update enters the AS with a certain metric, that metric is used to make decisions inside the AS. When the same update passes on to a third AS, that metric returns to 0. The diagram in this section shows the set of metric. The metric default value is 0.

Unless a router receives other directions, the router compares metrics for paths from neighbors in the same AS. In order for the router to compare metrics from neighbors that come from different ASs, you need to issue the special configuration command [bgp always-compare-med](#) on the router.

Note: There are two BGP configuration commands that can influence the multi-exit discriminator (MED)-based path selection. The commands are the [bgp deterministic-med](#) command and the [bgp always-compare-med](#) command. An issue of the [bgp deterministic-med](#) command ensures the comparison of the MED variable at route choice when different peers advertise in the same AS. An issue of the [bgp always-compare-med](#) command ensures the comparison of the MED for paths from neighbors in different ASs. The [bgp always-compare-med](#) command is

useful when multiple service providers or enterprises agree on a uniform policy for how to set MED. Refer to [How the bgp deterministic-med Command Differs from the bgp always-compare-med Command](#) to understand how these commands influence BGP path selection.

In the diagram in this section, AS100 gets information about network 180.10.0.0 via three different routers: RTC, RTD, and RTB. RTC and RTD are in AS300, and RTB is in AS400.

In this example, the AS-Path comparison on RTA by command [bgp bestpath as-path ignore](#) is ignored. It is configured to force BGP to fall on to the next attribute for route comparison (in this case metric or MED). If the command is omitted, the BGP will install route 180.10.0.0 from router RTC as that has the shortest AS-Path.

Assume that you have set the metric that comes from RTC to 120, the metric that comes from RTD to 200, and the metric that comes from RTB to 50. By default, a router compares metrics that come from neighbors in the same AS. Therefore, RTA can only compare the metric that comes from RTC to the metric that comes from RTD. RTA chooses RTC as the best next hop because 120 is less than 200. When RTA gets an update from RTB with metric 50, RTA cannot compare the metric to 120 because RTC and RTB are in different ASs. RTA must choose based on some other attributes.

In order to force RTA to compare the metrics, you must issue the **bgp always-compare-med** command on RTA. These configurations illustrate this process:

RTA#

```
router bgp 100
neighbor 2.2.2.1 remote-as 300
neighbor 3.3.3.3 remote-as 300
neighbor 4.4.4.3 remote-as 400
bgp bestpath as-path ignore
....
```

RTC#

```
router bgp 300
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 route-map setmetricout out
neighbor 1.1.1.2 remote-as 300
```

```
route-map setmetricout permit 10
set metric 120
```

RTD#

```
router bgp 300
neighbor 3.3.3.2 remote-as 100
neighbor 3.3.3.2 route-map setmetricout out
neighbor 1.1.1.1 remote-as 300
```

```
route-map setmetricout permit 10
set metric 200
```

RTB#

```
router bgp 400
neighbor 4.4.4.4 remote-as 100
neighbor 4.4.4.4 route-map setmetricout out
```

```
route-map setmetricout permit 10
  set metric 50
```

With these configurations, RTA picks RTC as next hop, with consideration of the fact that all other attributes are the same. In order to include RTB in the metric comparison, you must configure RTA in this way:

```
RTA#
router bgp 100
neighbor 2.2.21 remote-as 300
neighbor 3.3.3.3 remote-as 300
neighbor 4.4.4.3 remote-as 400
bgp always-compare-med
```

In this case, RTA picks RTB as the best next hop in order to reach network 180.10.0.0.

You can also set metric during the redistribution of routes into BGP if you issue the **default-metric number** command.

Assume that, in the example in this section, RTB injects a network via static into AS100. Here is the configuration:

```
RTB#
router bgp 400
redistribute static
default-metric 50
```

```
ip route 180.10.0.0 255.255.0.0 null 0
```

!--- This causes RTB to send out 180.10.0.0 with a metric of 50.

Community Attribute

The community attribute is a transitive, optional attribute in the range of 0 to 4,294,967,200. The community attribute is a way to group destinations in a certain community and apply routing decisions according to those communities. The routing decisions are accept, prefer, and redistribute, among others.

You can use route maps to set the community attributes. The route map **set** command has this syntax:

```
set community community-number [additive] [well-known-community]
```

A few predefined, well known communities for use in this command are:

no-export—Do not advertise to eBGP peers. Keep this route within an AS.

no-advertise—Do not advertise this route to any peer, internal or external.

internet—Advertise this route to the Internet community. Any router belongs to this community.

local-as—Use in confederation scenarios to prevent the transmit of packets outside the local

AS.

Here are two examples of route maps that set the community:

- route-map communitymap
match ip address 1
set community no-advertise

or

- route-map setcommunity
match as-path 1
set community 200 additive

If you do not set the **additive** keyword, 200 replaces any old community that already exists. If you use the keyword **additive**, an addition of 200 to the community occurs. Even if you set the community attribute, this attribute does not transmit to neighbors by default. In order to send the attribute to a neighbor, you must use this command:

```
neighbor {ip-address | peer-group-name} send-community
```

Here is an example:

```
RTA#  
router bgp 100  
neighbor 3.3.3.3 remote-as 300  
neighbor 3.3.3.3 send-community  
neighbor 3.3.3.3 route-map setcommunity out
```

In Cisco IOS Software Release 12.0 and later, you can configure communities in three different formats: decimal, hexadecimal, and AA:NN. By default, Cisco IOS Software uses the older decimal format. In order to configure and display in AA:NN, issue the **ip bgp-community new-format** global configuration command. The first part of AA:NN represents the AS number, and the second part represents a 2-byte number.

Here is an example:

Without the [ip bgp-community new-format](#) command in global configuration, an issue of the **show ip bgp 6.0.0.0** command displays the community attribute value in decimal format. In this example, the community attribute value appears as 6553620.

```
Router# show ip bgp 6.0.0.0 BGP routing table entry for 6.0.0.0/8, version 7 Paths: (1  
available, best #1, table Default-IP-Routing-Table) Not advertised to any peer 1 10.10.10.1 from  
10.10.10.1 (200.200.200.1) Origin IGP, metric 0, localpref 100, valid, external, best  
Community: 6553620
```

Now, issue the **ip bgp-community new-format** command globally on this router.

```
Router# configure terminal Enter configuration commands, one per line. End with CNTL/Z.
```

```
Router(config)# ip bgp-community new-format Router(config)# exit
```

With the **ip bgp-community new-format** global configuration command, the community value displays in AA:NN format. The value appears as 100:20 in the output of the **show ip bgp 6.0.0.0** command in this example:

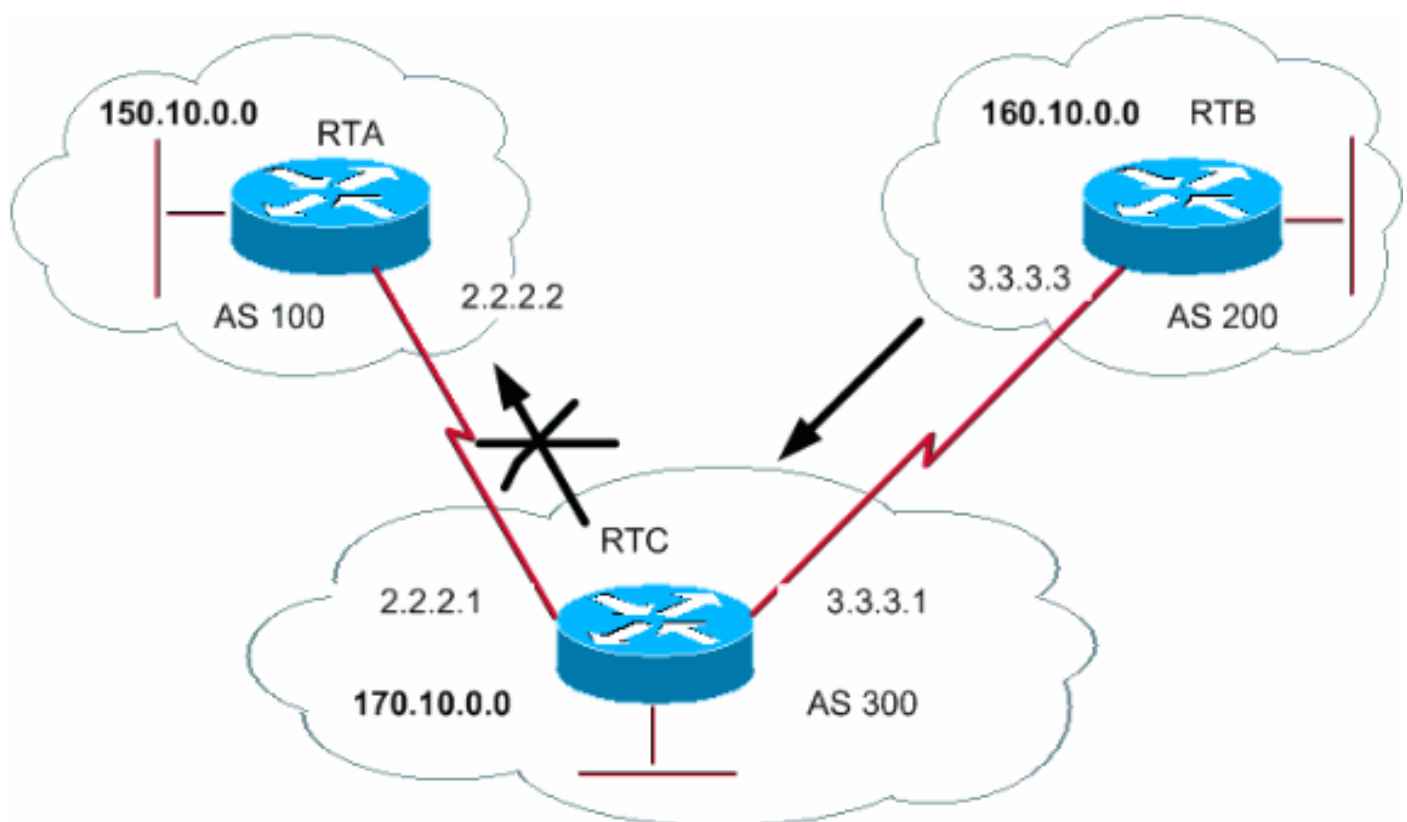
```
Router# show ip bgp 6.0.0.0 BGP routing table entry for 6.0.0.0/8, version 9 Paths: (1
available, best #1, table Default-IP-Routing-Table) Not advertised to any peer 1 10.10.10.1 from
10.10.10.1 (200.200.200.1) Origin IGP, metric 0, localpref 100, valid, external, best
Community: 100:20
```

BGP Case Studies 3

BGP Filtering

A number of different filter methods allow you to control the send and receive of BGP updates. You can filter BGP updates with route information as a basis, or with path information or communities as a basis. All methods achieve the same results. The choice of one method over another method depends on the specific network configuration.

Route Filtering



In order to restrict the routing information that the router learns or advertises, you can filter BGP with the use of routing updates to or from a particular neighbor. You define an access list and apply the access list to the updates to or from a neighbor. Issue this command in the router configuration mode:

```
neighbor {ip-address | peer-group-name} distribute-list access-list-number {in | out}
```

In this example, RTB originates network 160.10.0.0 and sends the update to RTC. If RTC wants to stop the propagation of the updates to AS100, you must define an access list to filter those updates and apply the access list during communication with RTA:

```
RTC#
router bgp 300
network 170.10.0.0
```

```
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 distribute-list 1 out

access-list 1 deny 160.10.0.0 0.0.255.255
```

```
access-list 1 permit 0.0.0.0 255.255.255.255
!--- Filter out all routing updates about 160.10.x.x.
```

The use of access lists is a bit tricky when you deal with supernets that can cause some conflicts.

Assume that, in the example in this section, RTB has different subnets of 160.10.x.x. Your goal is to filter updates and advertise only 160.0.0.0/8.

Note: The /8 notation means that you use 8 bits of subnet mask, which start from the far left of the IP address. This address is equivalent to 160.0.0.0 255.0.0.0.

The command **access-list 1 permit 160.0.0.0 0.255.255.255** permits 160.0.0.0/8, 160.0.0.0/9, and so on. In order to restrict the update to only 160.0.0.0/8, you must use an extended access list of this format:

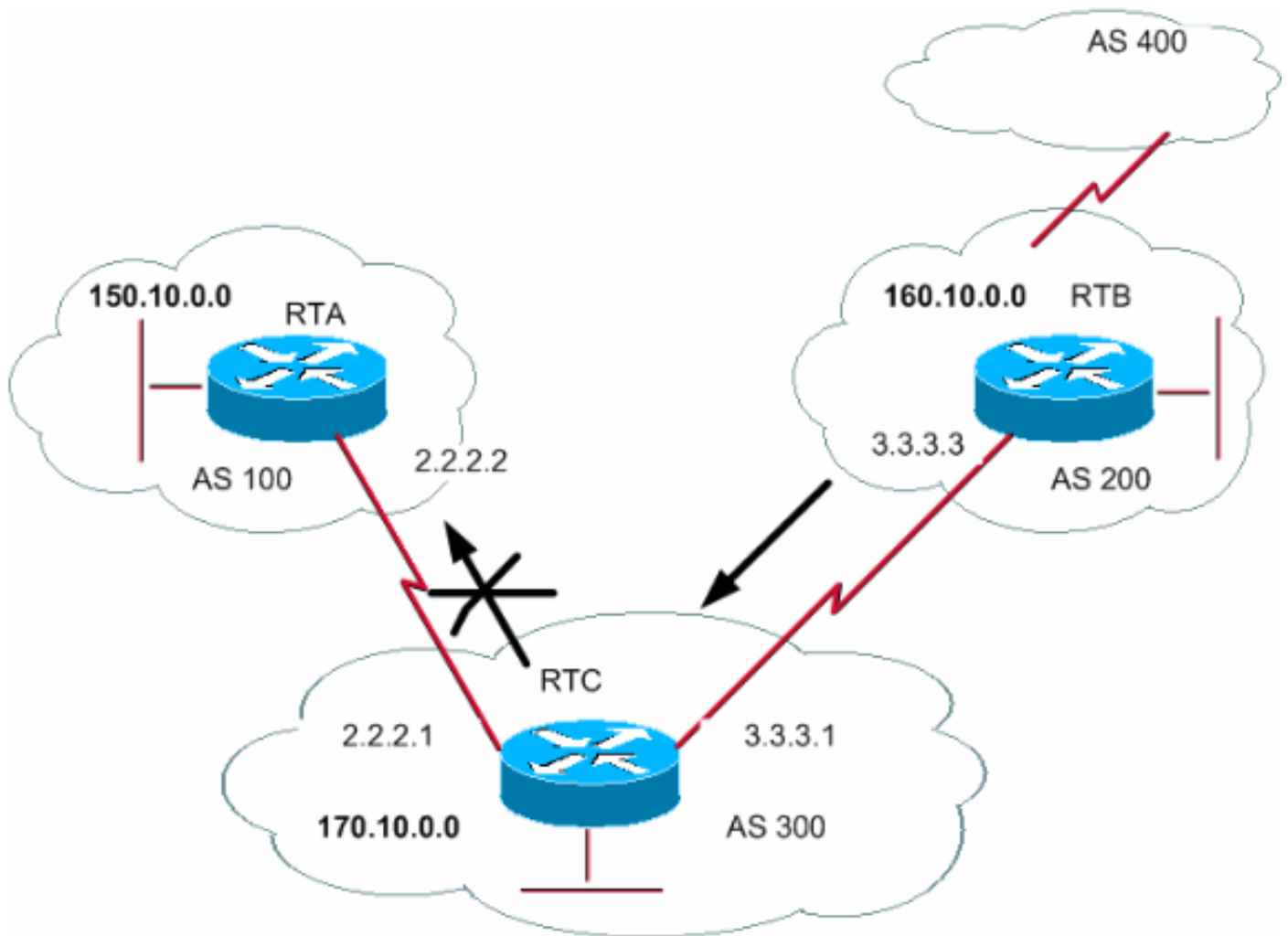
```
access-list 101 permit ip 160.0.0.0 0.255.255.255 255.0.0.0 0.0.0.0.
```

This list permits 160.0.0.0/8 only.

Refer to [How to Block One or More Networks From a BGP Peer](#) for sample configurations on how to filter networks from BGP peers. The method uses the **distribute-list** command with standard and extended access control lists (ACLs), as well as prefix list filtering.

[Path Filtering](#)

Another type of filtering is path filtering.



You can specify an access list on both incoming and outgoing updates with use of the BGP AS paths information. In the diagram in this section, you can block updates about 160.10.0.0 so that they do not go to AS100. To block the updates, define an access list on RTC that prevents the transmit to AS100 of any updates that have originated from AS200. Issue these commands:

```
ip as-path access-list access-list-number {permit | deny} as-regular-expression
neighbor {ip-address | peer-group-name} filter-list access-list-number {in | out}
```

This example stops the RTC send of updates about 160.10.0.0 to RTA:

```
RTC#
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 filter-list 1 out
!--- The 1 is the access list number below. ip as-path access-list 1 deny ^200$ ip as-path
access-list 1 permit .*
```

The **access-list 1** command in this example forces the denial of any updates with path information that starts with 200 and ends with 200. The `^200$` in the command is a "regular expression", in which `^` means "starts with" and `$` means "ends with". Since RTB sends updates about 160.10.0.0 with path information that starts with 200 and ends with 200, the updates match the access list. The access list denies these updates.

The `.*` is another regular expression in which the `.` means "any character" and the `*` means "the repetition of that character". So `.*` represents any path information, which is necessary to permit

the transmission of all other updates.

What happens if, instead of the use of `^200$`, you use `^200`? With an AS400, as in the diagram in this section, updates that AS400 originates have path information of the form (200, 400). In this path information, 200 is first and 400 is last. These updates match the access list `^200` because the path information starts with 200. The access list prevents the transmission of these updates to RTA, which is not the requirement.

In order to check if you have implemented the correct regular expression, issue the [show ip bgp regexp regular-expression](#) command. This command shows all the paths that have matched the regular expression configuration.

[AS Regular Expression](#)

This section explains the creation of a regular expression.

A regular expression is a pattern to match against an input string. When you build a regular expression, you specify a string that input must match. In the case of BGP, you specify a string that consists of path information that an input must match.

In the example in the section [Path Filtering](#), you specified the string `^200$`. You wanted path information that comes inside updates to match the string in order to make a decision.

A regular expression comprises:

Range

A range is a sequence of characters within left and right square brackets. An example is `[abcd]`.

Atom

An atom is a single character. Here are some examples:

.

The `.` matches any single character.

^

The `^` matches the start of the input string.

\$

The `$` matches the end of the input string.

\

The `\` matches the character.

_

The `_` matches a comma (`,`), left brace (`{`), right brace (`}`), the start of the input string, the

end of the input string, or a space.

Piece

A piece is one of these symbols, which follows an atom:

*

The * matches 0 or more sequences of the atom.

+

The + matches 1 or more sequences of the atom.

?

The ? matches the atom or the null string.

Branch

A branch is 0 or more concatenated pieces.

Here are some examples of regular expressions:

a*

This expression indicates any occurrence of the letter "a", which includes none.

a+

This expression indicates that at least one occurrence of the letter "a" must be present.

ab?a

This expression matches "aa" or "aba".

100

This expression means via AS100.

_100\$

This expression indicates an origin of AS100.

^100 .*

This expression indicates transmission from AS100.

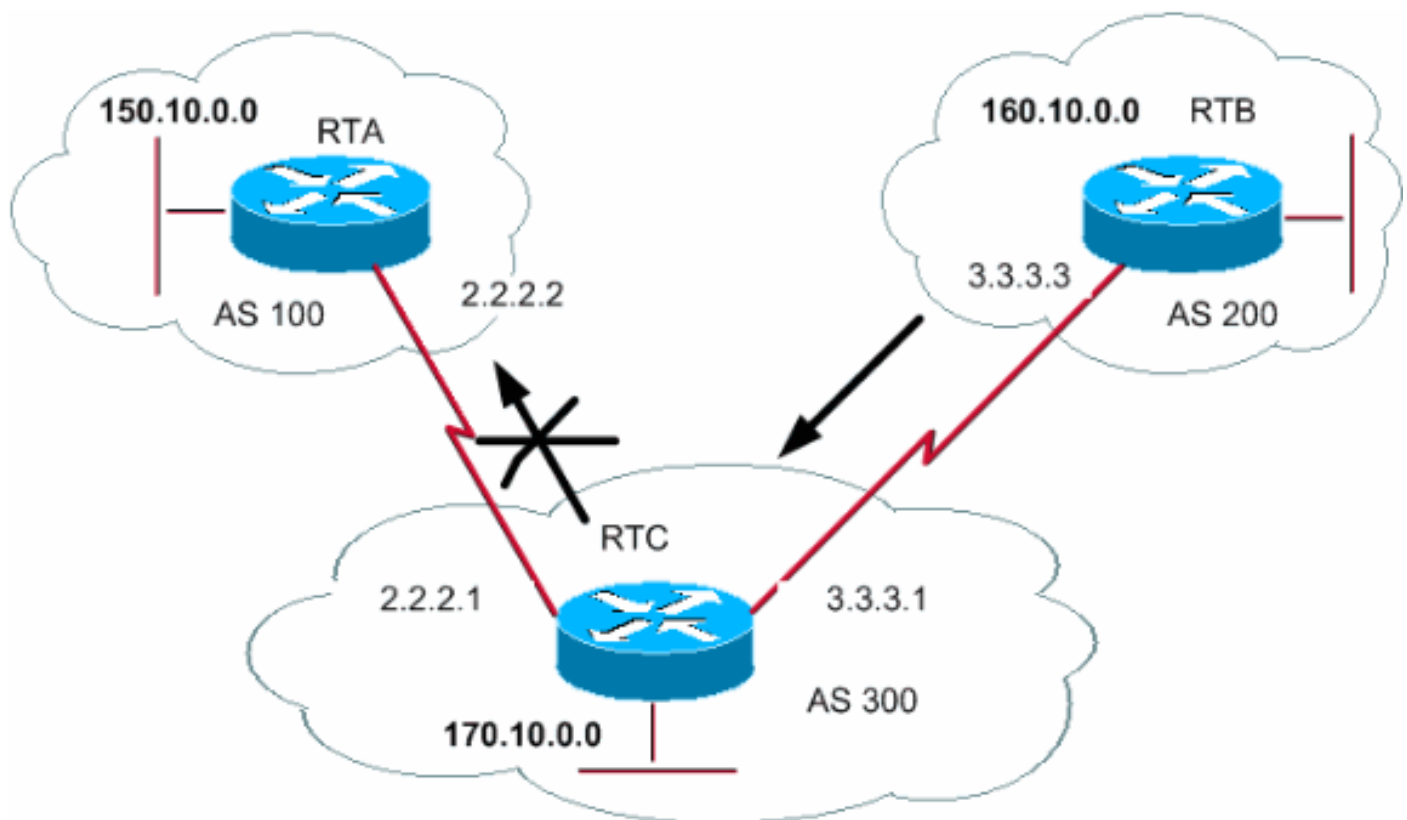
^\$

This expression indicates origination from this AS.

Refer to [Using Regular Expressions in BGP](#) for sample configurations of regular expression filtering.

BGP Community Filtering

This document has covered route filtering and AS-path filtering. Another method is community filtering. The section [Community Attribute](#) discusses community, and this section provides a few examples of how to use community.



In this example, you want RTB to set the community attribute to the BGP routes that RTB advertises such that RTC does not propagate these routes to the external peers. Use the **no-export** community attribute.

```
RTB#
router bgp 200
network 160.10.0.0
neighbor 3.3.3.1 remote-as 300
neighbor 3.3.3.1 send-community
neighbor 3.3.3.1 route-map setcommunity out

route-map setcommunity
match ip address 1
set community no-export

access-list 1 permit 0.0.0.0 255.255.255.255
```

Note: This example uses the **route-map setcommunity** command in order to set the community to **no-export**.

Note: The **neighbor send-community** command is necessary in order to send this attribute to RTC.

When RTC gets the updates with the attribute NO_EXPORT, RTC does not propagate the updates to external peer RTA.

In this example, RTB has set the community attribute to **100 200 additive**. This action adds the value 100 200 to any existing community value before transmission to RTC.

```
RTB#
router bgp 200
network 160.10.0.0
neighbor 3.3.3.1 remote-as 300
neighbor 3.3.3.1 send-community
neighbor 3.3.3.1 route-map setcommunity out
```

```
route-map setcommunity
match ip address 2
set community 100 200 additive
```

```
access-list 2 permit 0.0.0.0 255.255.255.255
```

A community list is a group of communities that you use in a **match** clause of a route map. The community list allows you to filter or set attributes with different lists of community numbers as a basis.

```
ip community-list community-list-number {permit | deny} community-number
```

For example, you can define this route map, **match-on-community**:

```
route-map match-on-community
match community 10
!--- The community list number is 10. set weight 20 ip community-list 10 permit 200 300 !--- The
community number is 200 300.
```

You can use the community list in order to filter or set certain parameters, like weight and metric, in certain updates with the community value as a basis. In the second example in this section, RTB sent updates to RTC with a community of 100 200. If RTC wants to set the weight with those values as a basis, you can do this:

```
RTC#
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 3.3.3.3 route-map check-community in
```

```
route-map check-community permit 10
match community 1
set weight 20
```

```
route-map check-community permit 20
match community 2 exact
set weight 10
```

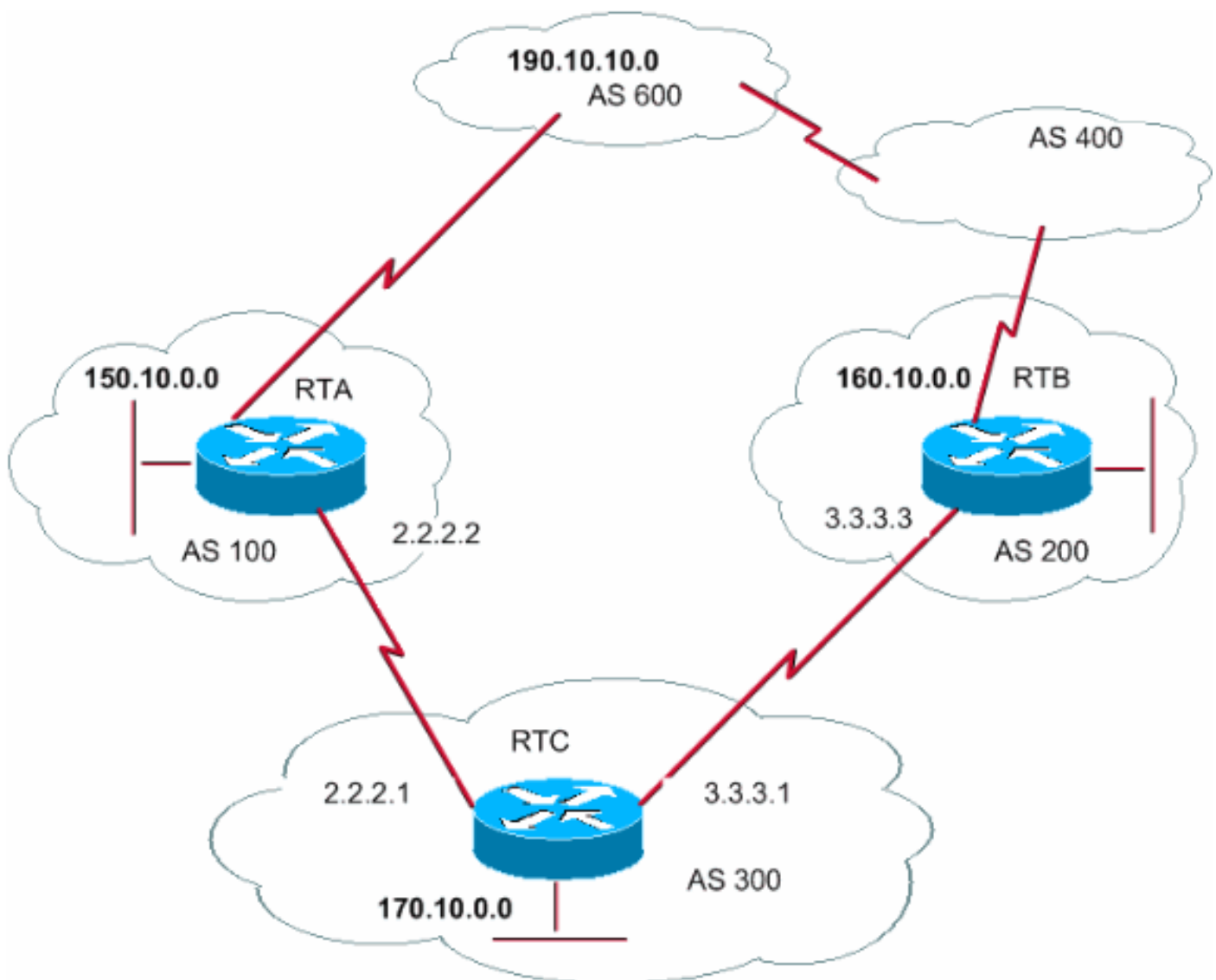
```
route-map check-community permit 30
match community 3
```

```
ip community-list 1 permit 100
ip community-list 2 permit 200
ip community-list 3 permit internet
```

In this example, any route that has 100 in the community attribute matches list 1. The weight of this route is set to 20. Any route that has only 200 as community matches list 2 and has a weight of 20. The keyword **exact** states that the community consists of 200 only and nothing else. The last community list is here to make sure that other updates do not drop. Remember that anything that does not match drops, by default. The keyword **internet** indicates all routes because all routes are members of the Internet community.

Refer to [Using BGP Community Values to Control Routing Policy in an Upstream Provider Network](#) for more information.

BGP Neighbors and Route Maps



You can use the **neighbor** command in conjunction with route maps to either filter or set parameters on incoming and outgoing updates.

Route maps associated with the **neighbor** statement have no effect on incoming updates when you match based on the IP address:

```
neighbor ip-address route-map route-map-name
```

Assume that, in the diagram in this section, you want RTC to learn from AS200 about networks that are local to AS200 and nothing else. Also, you want to set the weight on the accepted routes to 20. Use a combination of **neighbor** and **as-path** access lists:

```
RTC#
  router bgp 300
  network 170.10.0.0
  neighbor 3.3.3.3 remote-as 200
  neighbor 3.3.3.3 route-map stamp in

route-map stamp
match as-path 1
set weight 20

ip as-path access-list 1 permit ^200$
```

Any updates that originate from AS200 have path information that starts with 200 and ends with 200. These updates are permitted. Any other updates drop.

Assume that you want:

An acceptance of updates that originate from AS200 and have a weight of 20

The drop of updates that originate from AS400

A weight of 10 for other updates

```
RTC#
  router bgp 300
  network 170.10.0.0
  neighbor 3.3.3.3 remote-as 200
  neighbor 3.3.3.3 route-map stamp in

route-map stamp permit 10
match as-path 1
set weight 20

route-map stamp permit 20
match as-path 2
set weight 10

ip as-path access-list 1 permit ^200$
ip as-path access-list 2 permit ^200 600 .*
```

This statement sets a weight of 20 for updates that are local to AS200. The statement also sets a weight of 10 for updates that are behind AS400, and drops updates that come from AS400.

Use of set as-path prepend Command

In some situations, you must manipulate the path information in order to manipulate the BGP decision process. The command that you use with a route map is:

```
set as-path prepend as-path# as-path#
```

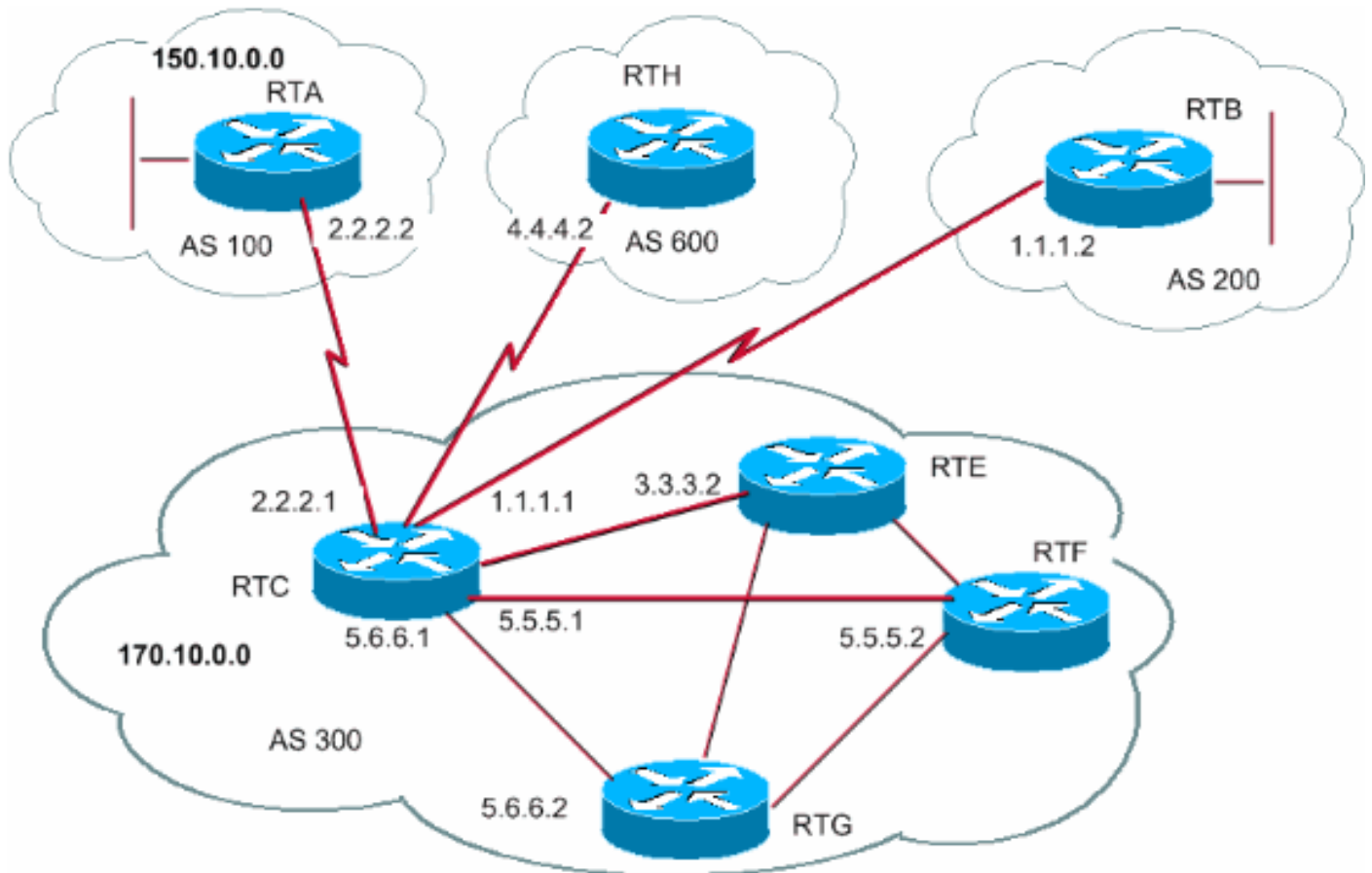
Suppose that, in the diagram in the section [BGP Neighbors and Route Maps](#), RTC advertises its own network 170.10.0.0 to two different ASs, AS100 and AS200. When the information is propagated to AS600, the routers in AS600 have network reachability information about 170.10.0.0 via two different routes. The first route is via AS100 with path (100, 300), and the second one is via AS400 with path (400, 200, 300). If all other attributes are the same, AS600 picks the shortest path and chooses the route via AS100.

AS300 gets all traffic via AS100. If you want to influence this decision from the AS300 end, you can make the path through AS100 appear to be longer than the path that goes through AS400. You can do this if you prepend AS numbers to the existing path information that is advertised to AS100. A common practice is to repeat your own AS number in this way:

```
RTC#  
router bgp 300  
network 170.10.0.0  
neighbor 2.2.2.2 remote-as 100  
neighbor 2.2.2.2 route-map SETPATH out  
  
route-map SETPATH  
set as-path prepend 300 300
```

Because of this configuration, AS600 receives updates about 170.10.0.0 via AS100 with path information of: (100, 300, 300, 300). This path information is longer than the (400, 200, 300) that AS600 received from AS400.

[BGP Peer Groups](#)



A BGP peer group is a group of BGP neighbors with the same update policies. Route maps, distribute lists, and filter lists typically set update policies. You do not define the same policies for each separate neighbor; instead, you define a peer group name and assign these policies to the peer group.

Members of the peer group inherit all the configuration options of the peer group. You can also configure members to override these options if the options do not affect outbound updates. You can only override options that are set on the inbound.

In order to define a peer group, issue this command:

```
neighbor peer-group-name peer-group
```

This example applies peer groups to internal and external BGP neighbors:

```
RTC#
  router bgp 300
  neighbor internalmap peer-group
  neighbor internalmap remote-as 300
  neighbor internalmap route-map SETMETRIC out
  neighbor internalmap filter-list 1 out
  neighbor internalmap filter-list 2 in
  neighbor 5.5.5.2 peer-group internalmap
  neighbor 5.6.6.2 peer-group internalmap
  neighbor 3.3.3.2 peer-group internalmap
  neighbor 3.3.3.2 filter-list 3 in
```

This configuration defines a peer group with the name **internalmap**. The configuration defines some policies for the group, such as a route map **SETMETRIC** to set the metric to 5 and two different filter lists, 1 and 2. The configuration applies the peer group to all internal neighbors,

RTE, RTF, and RTG. Also, the configuration defines a separate filter list 3 for neighbor RTE. This filter list overrides filter list 2 inside the peer group.

Note: You can only override options that affect inbound updates.

Now, look at how you can use peer groups with external neighbors. With the same diagram in this section, you configure RTC with a peer group **externalmap** and apply the peer group to external neighbors.

```
RTC#
router bgp 300
neighbor externalmap peer-group
neighbor externalmap route-map SETMETRIC
neighbor externalmap filter-list 1 out
neighbor externalmap filter-list 2 in
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 peer-group externalmap
neighbor 4.4.4.2 remote-as 600
neighbor 4.4.4.2 peer-group externalmap
neighbor 1.1.1.2 remote-as 200
neighbor 1.1.1.2 peer-group externalmap
neighbor 1.1.1.2 filter-list 3 in
```

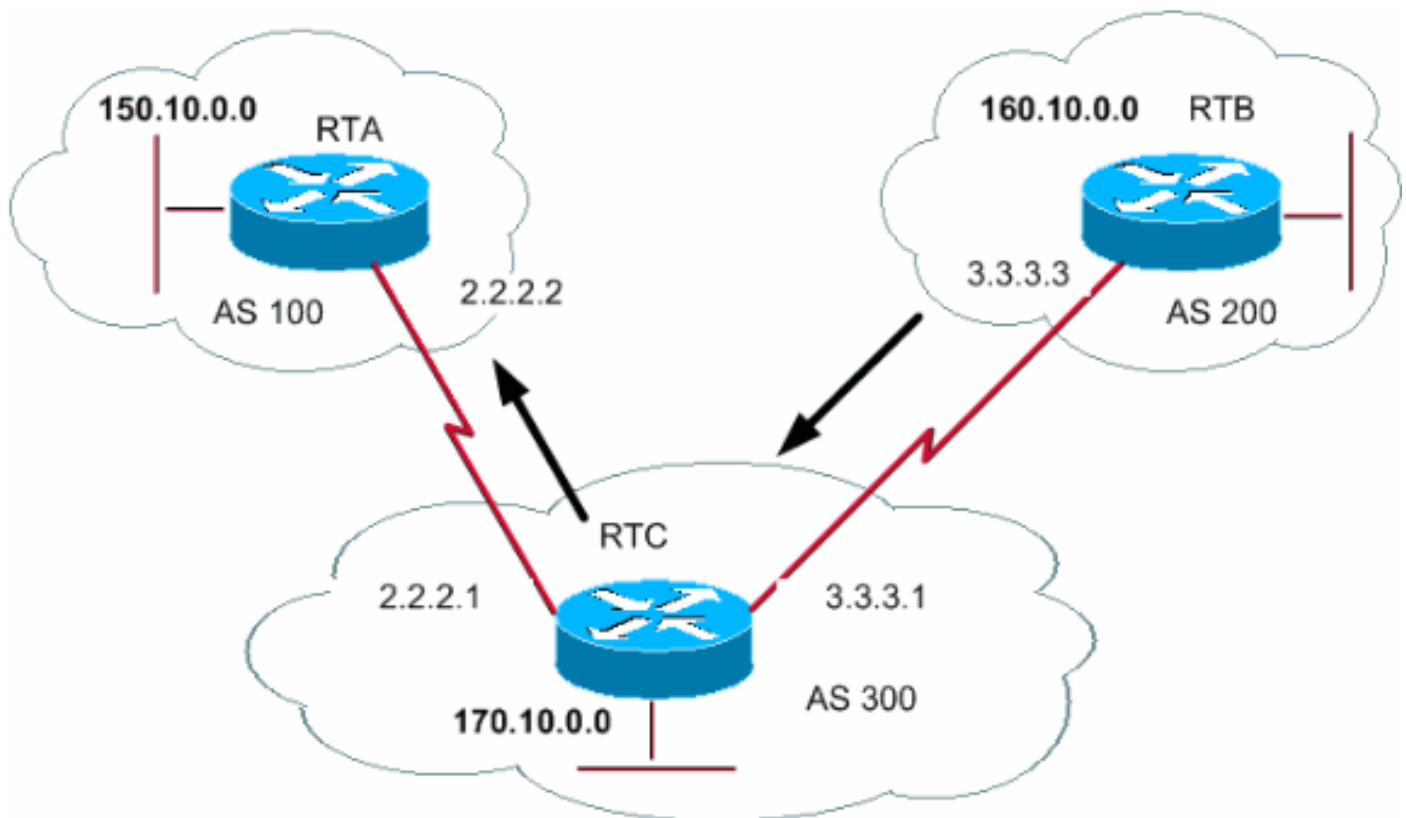
Note: In these configurations, you define the **remote-as** statements outside of the peer group because you must define different external ASs. Also, you override the inbound updates of neighbor 1.1.1.2 with the assignment of filter list 3.

For more information on peer groups, refer to [BGP Peer Groups](#).

Note: In Cisco IOS Software Release 12.0(24)S, Cisco introduced the BGP Dynamic Update Peer Groups feature. The feature is available in later Cisco IOS Software releases as well. The feature introduces a new algorithm that dynamically calculates and optimizes update groups of neighbors that share the same outbound policies. These neighbors can share the same update messages. In earlier releases of Cisco IOS Software, the group of BGP update messages was on the basis of peer group configurations. This method to group updates limited outbound policies and specific session configurations. The BGP Dynamic Update Peer Group feature separates update group replication from peer group configuration. This separation improves the convergence time and the flexibility of neighbor configuration. Refer to [BGP Dynamic Update Peer-Groups](#) for more details.

[BGP Case Studies 4](#)

[CIDR and Aggregate Addresses](#)



One of the main enhancements of BGP4 over BGP3 is classless interdomain routing (CIDR). CIDR or supernetting is a new way to look at IP addresses. With CIDR, there is no notion of classes, such as class A, B, or C. For example, network 192.213.0.0 was once an illegal class C network. Now, the network is a legal supernet, 192.213.0.0/16. The "16" represents the number of bits in the subnet mask, when you count from the far left of the IP address. This representation is similar to 192.213.0.0 255.255.0.0.

You use aggregates in order to minimize the size of routing tables. Aggregation is the process that combines the characteristics of several different routes in such a way that advertisement of a single route is possible. In this example, RTB generates network 160.10.0.0. You configure RTC to propagate a supernet of that route 160.0.0.0 to RTA:

```
RTB#
router bgp 200
neighbor 3.3.3.1 remote-as 300
network 160.10.0.0
```

```
#RTC
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
network 170.10.0.0
aggregate-address 160.0.0.0 255.0.0.0
```

RTC propagates the aggregate address 160.0.0.0 to RTA.

[Aggregate Commands](#)

There is a wide range of aggregate commands. You must understand how each one works in order to have the aggregation behavior that you desire.

The first command is the one from the example in the section [CIDR and Aggregate Addresses](#):

aggregate-address *address-mask*

This command advertises the prefix route and all the more-specific routes. The command **aggregate-address 160.0.0.0** propagates an additional network 160.0.0.0 but does not prevent the propagation of 160.10.0.0 to RTA. The outcome is the propagation of both networks 160.0.0.0 and 160.10.0.0 to RTA, which is the advertisement of both the prefix and the more-specific route.

Note: You cannot aggregate an address if you do not have a more-specific route of that address in the BGP routing table.

For example, RTB cannot generate an aggregate for 160.0.0.0 if RTB does not have a more-specific entry of 160.0.0.0 in the BGP table. An injection of the more-specific route into the BGP table is possible. The route injection can occur via:

Incoming updates from other ASs

Redistribution of an IGP or static into BGP

The **network** command, for example, **network 160.10.0.0**

If you want RTC to propagate network 160.0.0.0 only and **not** the more-specific route, issue this command:

aggregate-address *address mask summary-only*

This command advertises the prefix only. The command suppresses all the more-specific routes.

The command **aggregate 160.0.0.0 255.0.0.0 summary-only** propagates network 160.0.0.0 and suppresses the more-specific route 160.10.0.0.

Note: If you aggregate a network that injected into your BGP via the **network** statement, the network entry always injects into BGP updates. This injection occurs even though you use the **aggregate summary-only** command. The example in the section [CIDR Example 1](#) discusses this situation.

aggregate-address *address-mask as-set*

This command advertises the prefix and the more-specific routes. But the command includes **as-set** information in the path information of the routing updates.

aggregate 129.0.0.0 255.0.0.0 as-set

The section [CIDR Example 2 \(as-set\)](#) discusses this command.

If you want to suppress more-specific routes when you do the aggregation, define a route map and apply the route map to the aggregates. The action allows you to be selective about which more-specific routes to suppress.

aggregate-address *address-mask* **suppress-map** *map-name*

This command advertises the prefix and the more-specific routes. But the command suppresses advertisement with a route map basis. Suppose that, with the diagram in the section [CIDR and Aggregate Addresses](#), you want to aggregate 160.0.0.0, suppress the more-specific route 160.20.0.0, and allow the propagation of 160.10.0.0. Use this route map:

```
route-map CHECK permit 10
match ip address 1
```

```
access-list 1 permit 160.20.0.0 0.0.255.255
```

```
access-list 1 deny 0.0.0.0 255.255.255.255
```

By definition of the **suppress-map**, there is a suppression from the updates of any packets that the access list permits.

Then, apply the route map to the **aggregate** statement.

```
RTC#
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 remote-as 100
network 170.10.0.0
aggregate-address 160.0.0.0 255.0.0.0 suppress-map CHECK
```

Here is another variation:

aggregate-address *address-mask* **attribute-map** *map-name*

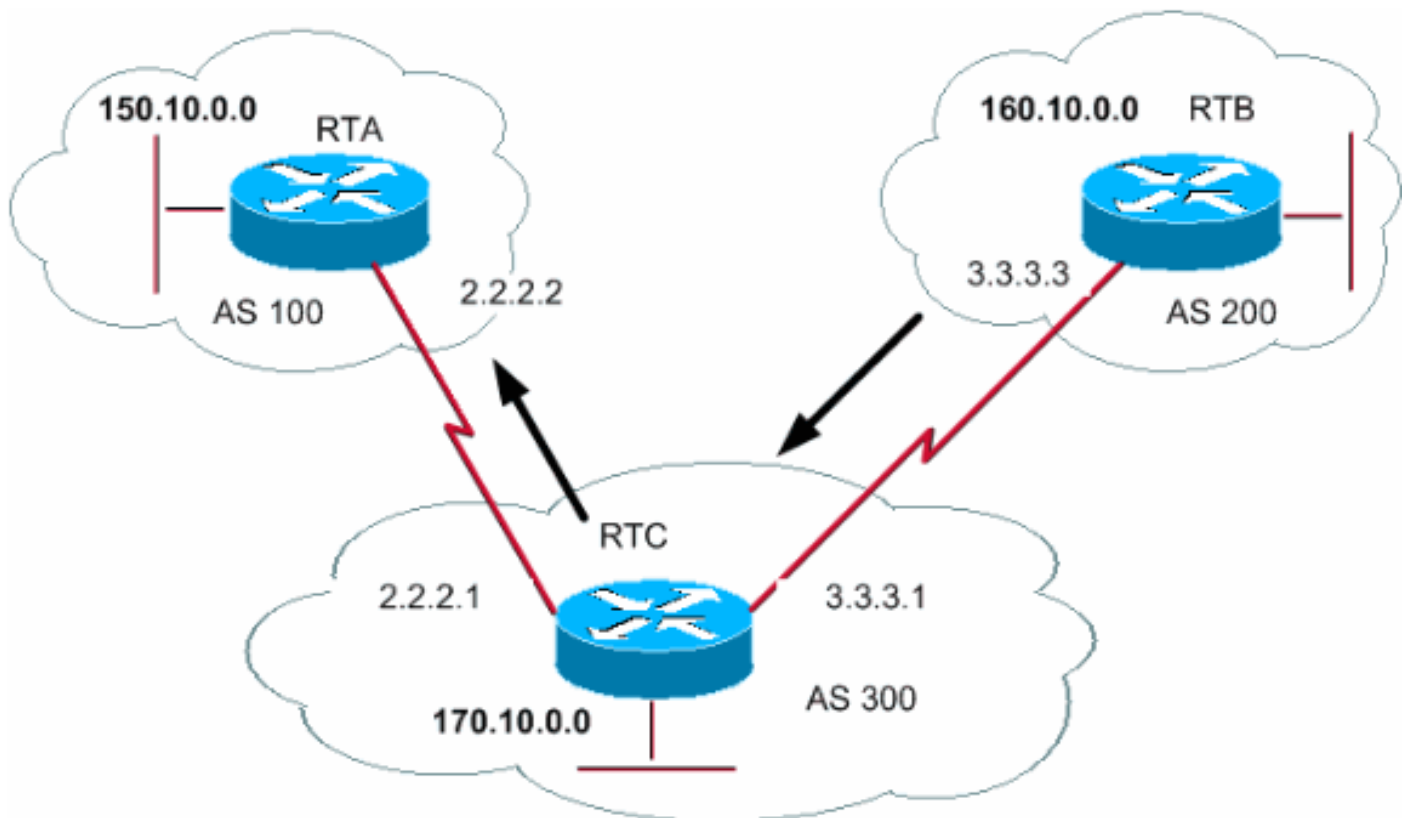
This command allows you to set the attributes, such as metric, at the time of the send of aggregates. In order to set the origin of the aggregates to IGP, apply this route map to the **aggregate attribute-map** command:

```
route-map SETMETRIC
set origin igp
```

```
aggregate-address 160.0.0.0 255.0.0.0 attribute-map SETORIGIN
```

For more information, refer to [Understanding Route Aggregation in BGP](#).

[CIDR Example 1](#)



Request: Allow RTB to advertise the prefix 160.0.0.0 and suppress all the more-specific routes. The problem with this request is that network 160.10.0.0 is local to AS200, which means that AS200 is the originator of 160.10.0.0. You cannot have RTB generate a prefix for 160.0.0.0 without the generation of an entry for 160.10.0.0, even if you use the **aggregate summary-only** command. RTB generates both networks because RTB is the originator of 160.10.0.0. There are two solutions to this problem.

The first solution is to use a static route and redistribute into BGP. The outcome is that RTB advertises the aggregate with an origin of incomplete (?).

```

RTB#
router bgp 200
neighbor 3.3.3.1 remote-as 300
redistribute static
!--- This generates an update for 160.0.0.0 !--- with the origin path as "incomplete". ip route
160.0.0.0 255.0.0.0 null0

```

In the second solution, in addition to the static route, you add an entry for the **network** command. This entry has the same effect, except that the entry sets the origin of the update to IGP.

```

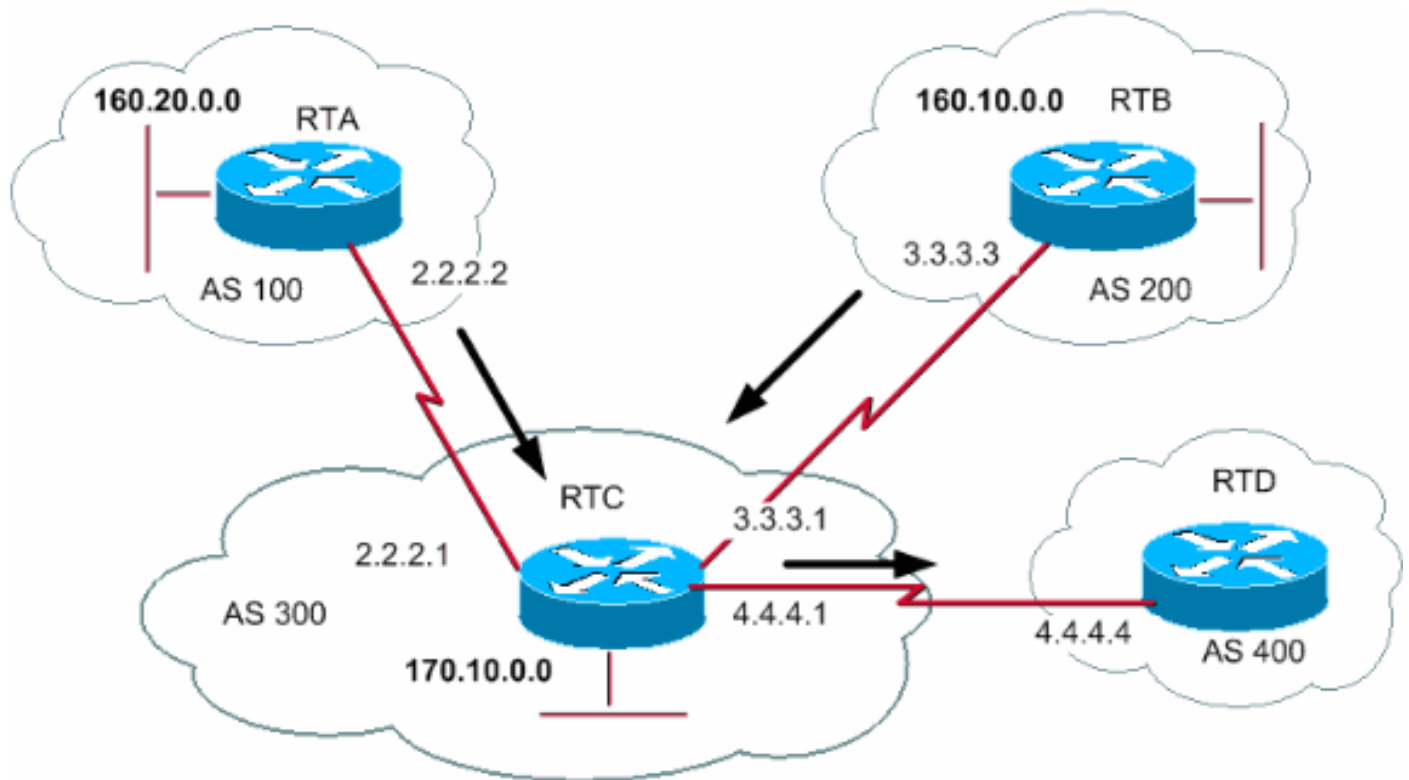
RTB#
router bgp 200
network 160.0.0.0 mask 255.0.0.0
!--- This entry marks the update with origin IGP. neighbor 3.3.3.1 remote-as 300 redistribute
static ip route 160.0.0.0 255.0.0.0 null0

```

[CIDR Example 2 \(as-set\)](#)

You use the statement **as-set** in aggregation to reduce the size of the path information. With **as-set**, the AS number is listed only once, regardless of how many times the AS number appeared in multiple paths that were aggregated. You use the **aggregate as-set** command in situations in which the aggregation of information causes loss of information with regard to the path attribute. In

this example, RTC gets updates about 160.20.0.0 from RTA and updates about 160.10.0.0 from RTB. Suppose that RTC wants to aggregate network 160.0.0.0/8 and send the network to RTD. RTD does not know the origin of that route. If you add the **aggregate as-set** statement, you force RTC to generate path information in the form of a set {}. That set includes all the path information, irrespective of which path came first.



```
RTB#
router bgp 200
network 160.10.0.0
neighbor 3.3.3.1 remote-as 300
```

```
RTA#
router bgp 100
network 160.20.0.0
neighbor 2.2.2.1 remote-as 300
```

Case 1:

RTC does not have an **as-set** statement. RTC sends an update 160.0.0.0/8 to RTD with path information (300), as if the route originated from AS300.

```
RTC#
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
neighbor 4.4.4.4 remote-as 400
aggregate 160.0.0.0 255.0.0.0 summary-only
```

!--- This command causes RTC to send RTD updates about 160.0.0.0/8 !--- with no indication that 160.0.0.0 actually comes from two different ASs. !--- This may create loops if RTD has an entry back into AS100 or AS200.

Case 2:

```
RTC#
```

```
router bgp 300
neighbor 3.3.3.3 remote-as 200
neighbor 2.2.2.2 remote-as 100
neighbor 4.4.4.4 remote-as 400
aggregate 160.0.0.0 255.0.0.0 summary-only
aggregate 160.0.0.0 255.0.0.0 as-set
```

!--- This command causes RTC to send RTD updates about 160.0.0.0/8 !--- with an indication that 160.0.0.0 belongs to a set {100 200}.

The next two subjects, [BGP Confederation](#) and [Route Reflectors](#), are for Internet service providers (ISPs) that want further control of the explosion of iBGP peering inside their ASs.

BGP Confederation

The implementation of BGP confederation reduces the iBGP mesh inside an AS. The trick is to divide an AS into multiple ASs and assign the whole group to a single confederation. Each AS alone has iBGP fully meshed and has connections to other ASs inside the confederation. Even though these ASs have eBGP peers to ASs within the confederation, the ASs exchange routing as if they used iBGP. In this way, the confederation preserves next hop, metric, and local preference information. To the outside world, the confederation appears to be a single AS.

In order to configure a BGP confederation, issue this command:

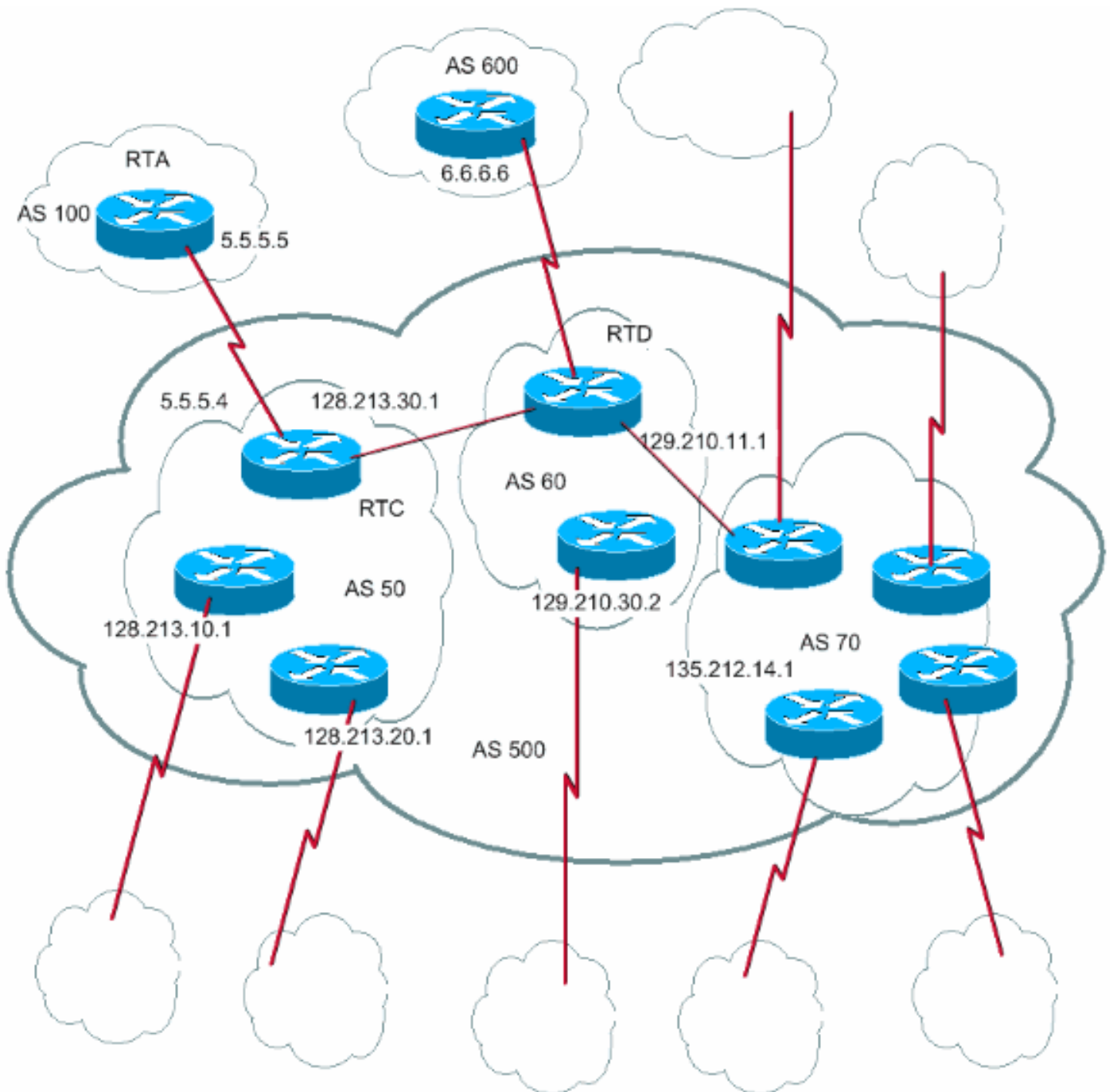
```
bgp confederation identifier autonomous-system
```

The confederation identifier is the AS number of the confederation group.

The issue of this command performs peering between multiple ASs within the confederation:

```
bgp confederation peers autonomous-system [autonomous-system]
```

Here is an example of confederation:



Assume that you have an AS500 that consists of nine BGP speakers. Other non-BGP speakers exist also, but you only have interest in the BGP speakers that have eBGP connections to other ASs. If you want to make a full iBGP mesh inside AS500, you need nine peer connections for each router. You need eight iBGP peers and one eBGP peer to external ASs.

If you use confederation, you can divide AS500 into multiple ASs: AS50, AS60, and AS70. You give the AS a confederation identifier of 500. The outside world sees only one AS, AS500. For each of AS50, AS60, and AS70, you define a full mesh of iBGP peers, and you define the list of confederation peers with the **bgp confederation peers** command.

Here is a sample configuration of routers RTC, RTD, and RTA:

Note: RTA has no knowledge of AS50, AS60, or AS70. RTA has only knowledge of AS500.

```
RTC#
router bgp 50
```

```
bgp confederation identifier 500
bgp confederation peers 60 70
neighbor 128.213.10.1 remote-as 50 (IBGP connection within AS50)
neighbor 128.213.20.1 remote-as 50 (IBGP connection within AS50)
neighbor 129.210.11.1 remote-as 60 (BGP connection with confederation peer 60)
neighbor 135.212.14.1 remote-as 70 (BGP connection with confederation peer 70)
neighbor 5.5.5.5 remote-as 100 (EBGP connection to external AS100)
```

RTD#

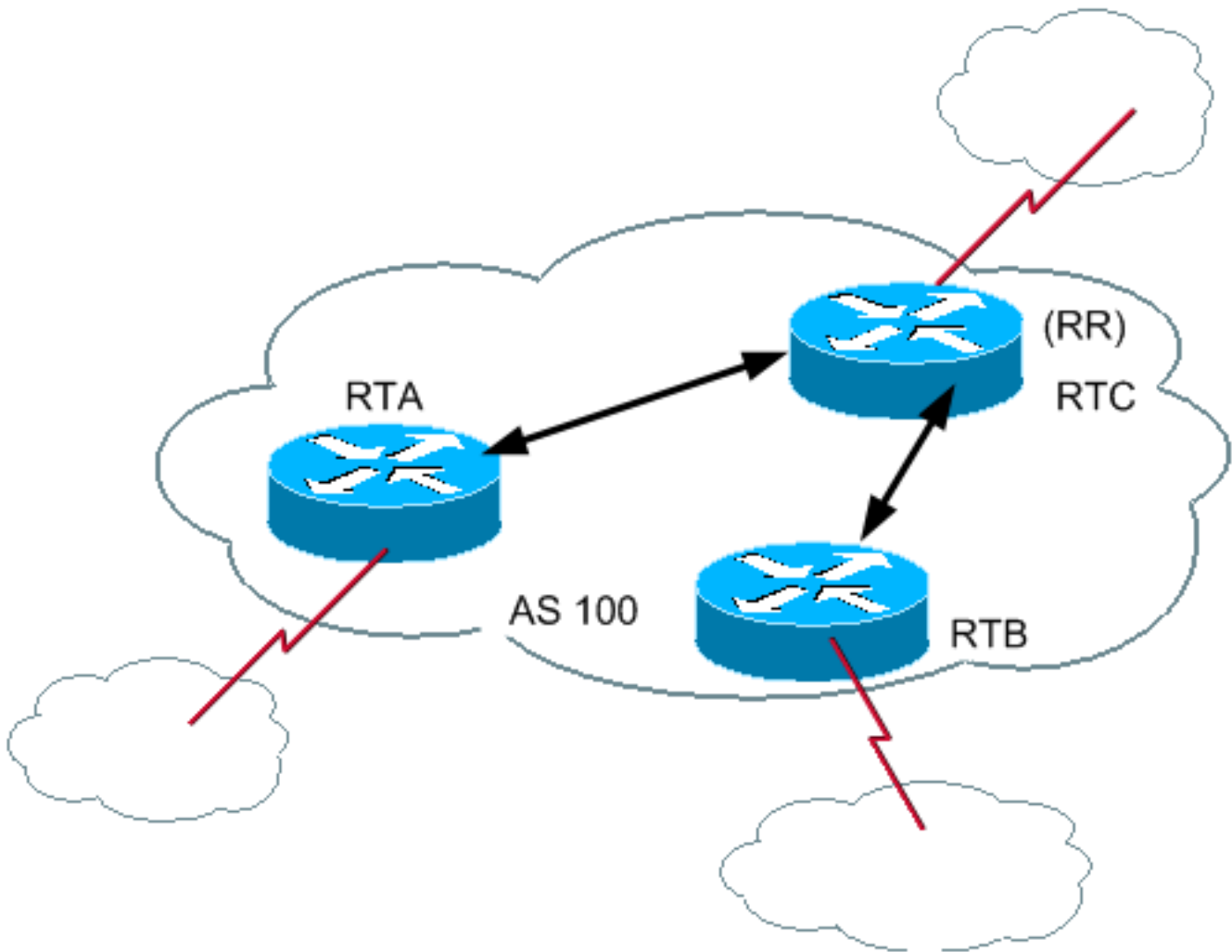
```
router bgp 60
bgp confederation identifier 500
bgp confederation peers 50 70
neighbor 129.210.30.2 remote-as 60 (IBGP connection within AS60)
neighbor 128.213.30.1 remote-as 50 (BGP connection with confederation peer 50)
neighbor 135.212.14.1 remote-as 70 (BGP connection with confederation peer 70)
neighbor 6.6.6.6 remote-as 600 (EBGP connection to external AS600)
```

RTA#

```
router bgp 100
neighbor 5.5.5.4 remote-as 500 (EBGP connection to confederation 500)
```

Route Reflectors

Another solution for the explosion of iBGP peering within an AS is Route Reflectors (RRs). As the [iBGP](#) section demonstrates, a BGP speaker does not advertise a route that the BGP speaker learned via another iBGP speaker to a third iBGP speaker. You can relax this restriction a bit and provide additional control, which allows a router to advertise, or reflect, iBGP learned routes to other iBGP speakers. This route reflection reduces the number of iBGP peers within an AS.

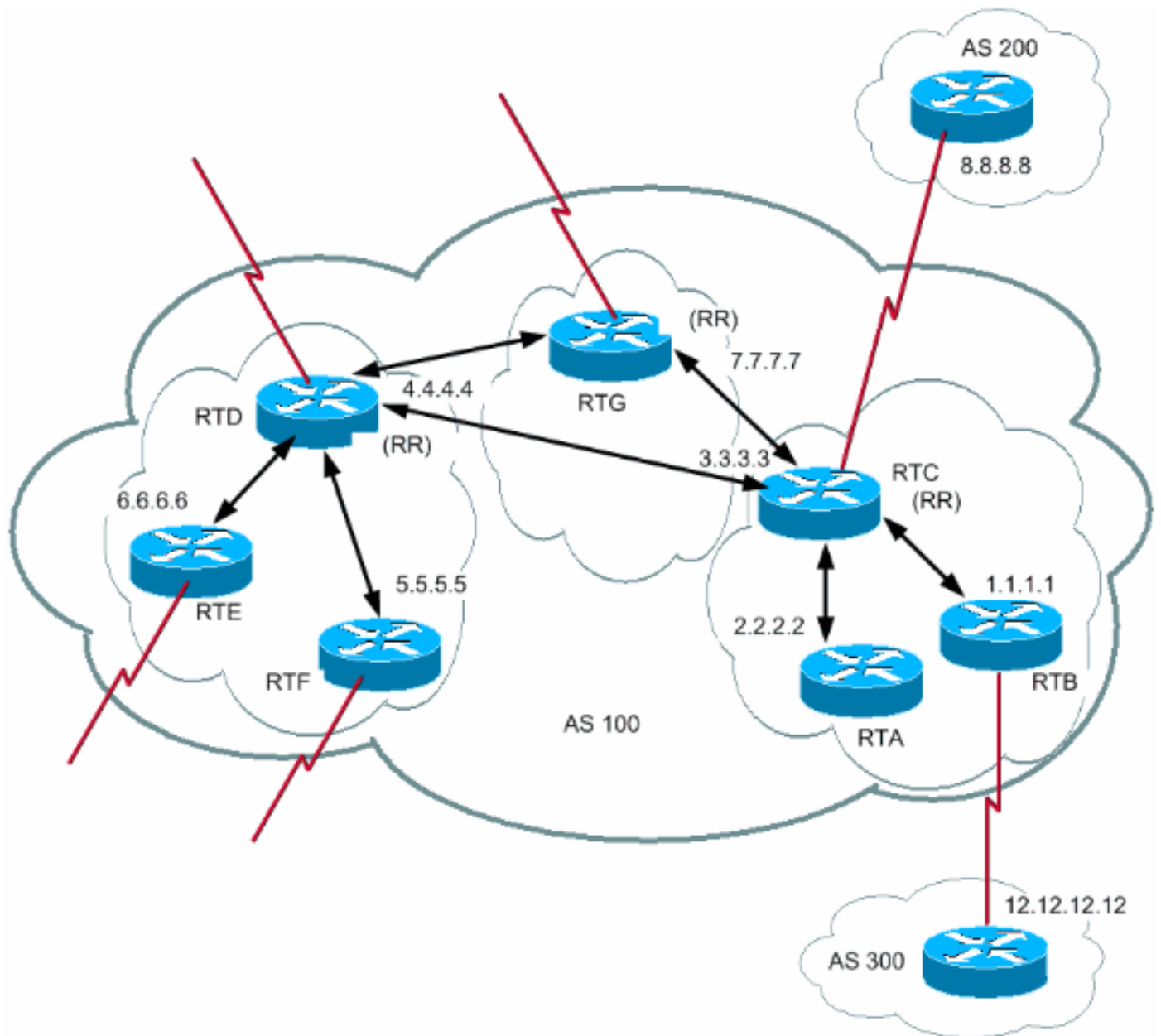


In normal cases, maintain a full iBGP mesh between RTA, RTB, and RTC within AS100. If you utilize the RR concept, RTC can be elected as an RR. In this way, RTC has a partial iBGP peering with RTA and RTB. Peering between RTA and RTB is not necessary because RTC is an RR for the updates that come from RTA and RTB.

[neighbor route-reflector-client](#)

The router with this command is the RR, and the neighbors at which the command points are the clients of that RR. In the example, the RTC configuration has the **neighbor route-reflector-client** command that points at the RTA and RTB IP addresses. The combination of the RR and the clients is a "cluster". In this example, RTA, RTB, and RTC form a cluster with a single RR within AS100.

Other iBGP peers of the RR that are not clients are "nonclients".



An AS can have more than one RR. In this situation, an RR treats other RRs just like any other iBGP speaker. Other RRs can belong to the same cluster (client group) or to other clusters. In a simple configuration, you can divide the AS into multiple clusters. You configure each RR with other RRs as nonclient peers in a fully meshed topology. Clients should not peer with iBGP speakers outside the client cluster.

Consider this [diagram](#). RTA, RTB, and RTC form a single cluster. RTC is the RR. For RTC, RTA and RTB are clients and anything else is a nonclient. Remember that the **neighbor route-reflector-client** command points at clients of an RR. The same RTD is the RR for clients RTE and RTF. RTG is an RR in a third cluster.

Note: RTD, RTC, and RTG are fully meshed, but routers within a cluster are not. When an RR receives a route, the RR routes as this list shows. However, this activity depends on the peer type:

Routes from a nonclient peer—Reflects to all the clients within the cluster.

Routes from a client peer—Reflects to all the nonclient peers and also to the client peers.

Routes from an eBGP peer—Sends the update to all client and nonclient peers.

Here is the relative BGP configuration of routers RTC, RTD, and RTB:

RTC#

```
router bgp 100
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 route-reflector-client
neighbor 1.1.1.1 remote-as 100
neighbor 1.1.1.1 route-reflector-client
neighbor 7.7.7.7 remote-as 100
neighbor 4.4.4.4 remote-as 100
neighbor 8.8.8.8 remote-as 200
```

RTB#

```
router bgp 100
neighbor 3.3.3.3 remote-as 100
neighbor 12.12.12.12 remote-as 300
```

RTD#

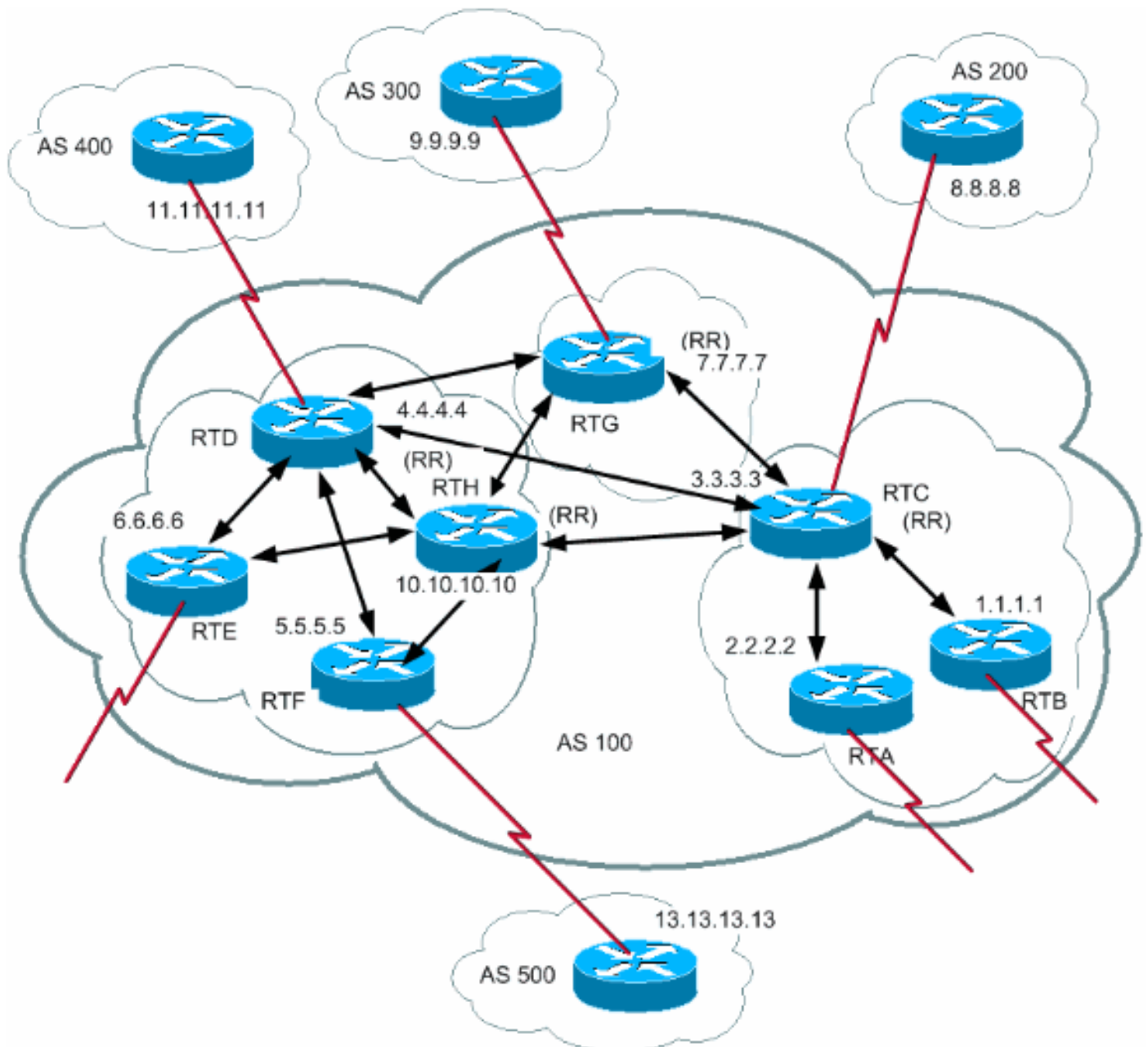
```
router bgp 100
neighbor 6.6.6.6 remote-as 100
neighbor 6.6.6.6 route-reflector-client
neighbor 5.5.5.5 remote-as 100
neighbor 5.5.5.5 route-reflector-client
neighbor 7.7.7.7 remote-as 100
neighbor 3.3.3.3 remote-as 100
```

Because there is a reflection of the iBGP learned routes, there can be a routing information loop. The RR scheme has a few methods to avoid this loop:

originator-id—This is an optional, nontransitive BGP attribute that is 4 bytes long. An RR creates this attribute. The attribute carries the router ID (RID) of the originator of the route in the local AS. If, due to poor configuration, the routing information comes back to the originator, the information is ignored.

cluster-list—The section [Multiple RRs within a Cluster](#) covers cluster list.

[Multiple RRs within a Cluster](#)



Usually, a cluster of clients has a single RR. In this case, the router ID of the RR identifies the cluster. In order to increase redundancy and avoid single points of failure, a cluster can have more than one RR. You need to configure all RRs in the same cluster with a 4-byte cluster ID so that an RR can recognize updates from RRs in the same cluster.

A cluster list is a sequence of cluster IDs that the route has passed. When an RR reflects a route from the RR clients to nonclients outside of the cluster, the RR appends the local cluster ID to the cluster list. If this update has an empty cluster list, the RR creates one. With this attribute, an RR can identify if the routing information has looped back to the same cluster due to poor configuration. If the local cluster ID is found in the cluster list, the advertisement is ignored.

In the diagram in this section, RTD, RTE, RTF, and RTH belong to one cluster. Both RTD and RTH are RRs for the same cluster.

Note: There is redundancy because RTH has fully meshed peering with all the RRs. If RTD goes down, RTH takes the place of RTD.

Here is the configuration of RTH, RTD, RTF, and RTC:

RTH#

```
router bgp 100
neighbor 4.4.4.4 remote-as 100
neighbor 5.5.5.5 remote-as 100
neighbor 5.5.5.5 route-reflector-client
neighbor 6.6.6.6 remote-as 100
neighbor 6.6.6.6 route-reflector-client
neighbor 7.7.7.7 remote-as 100
neighbor 3.3.3.3 remote-as 100
neighbor 9.9.9.9 remote-as 300
bgp cluster-id 10
```

RTD#

```
router bgp 100
neighbor 10.10.10.10 remote-as 100
neighbor 5.5.5.5 remote-as 100
neighbor 5.5.5.5 route-reflector-client
neighbor 6.6.6.6 remote-as 100
neighbor 6.6.6.6 route-reflector-client
neighbor 7.7.7.7 remote-as 100
neighbor 3.3.3.3 remote-as 100
neighbor 11.11.11.11 remote-as 400
bgp cluster-id 10
```

RTF#

```
router bgp 100
neighbor 10.10.10.10 remote-as 100
neighbor 4.4.4.4 remote-as 100
neighbor 13.13.13.13 remote-as 500
```

RTC#

```
router bgp 100
neighbor 1.1.1.1 remote-as 100
neighbor 1.1.1.1 route-reflector-client
neighbor 2.2.2.2 remote-as 100
neighbor 2.2.2.2 route-reflector-client
neighbor 4.4.4.4 remote-as 100
neighbor 7.7.7.7 remote-as 100
neighbor 10.10.10.10 remote-as 100
neighbor 8.8.8.8 remote-as 200
```

Note: You do not need the [bgp cluster-id](#) command for RTC because only one RR exists in that cluster.

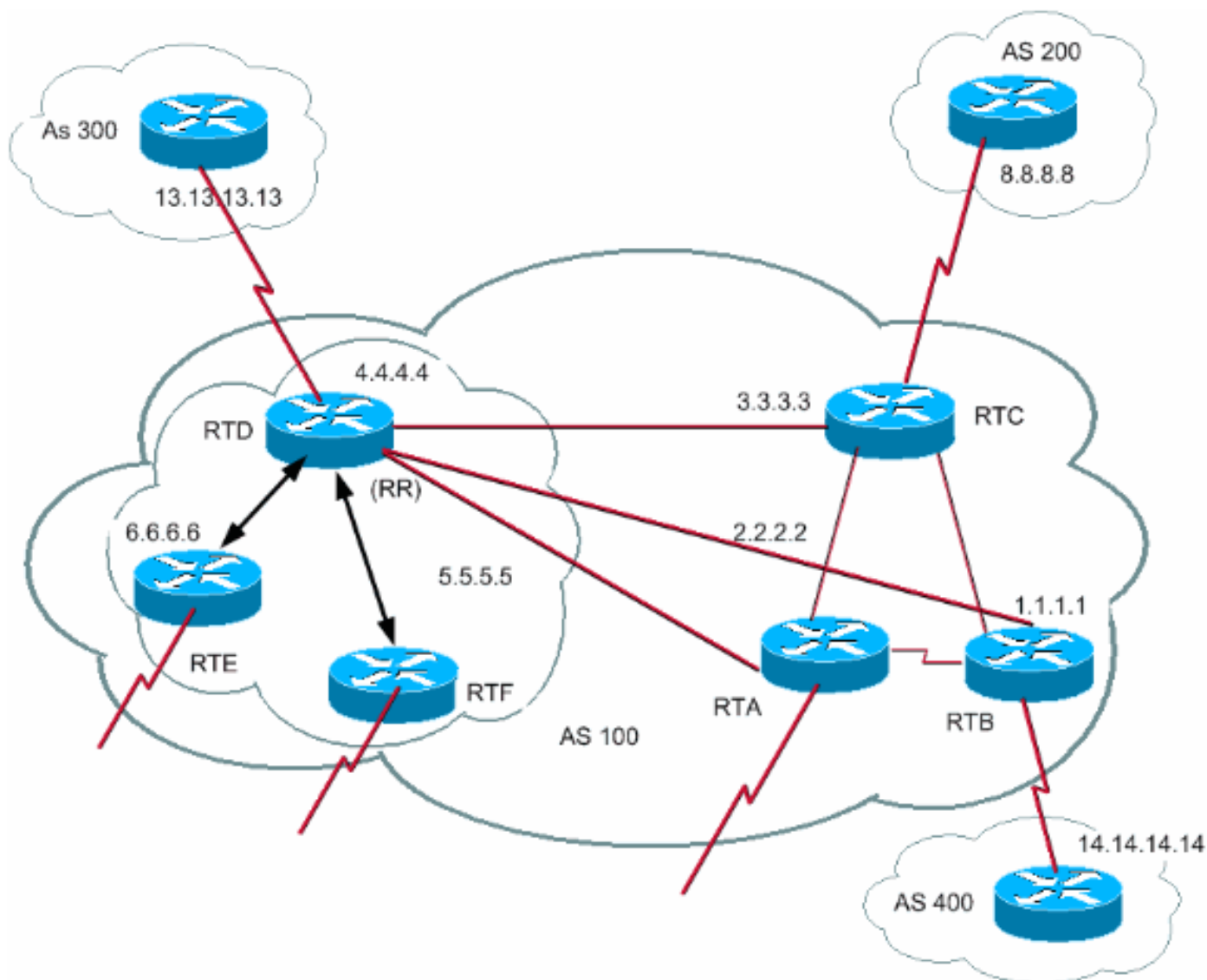
Important Note: This configuration does not use peer groups. Do not use peer groups if the clients inside a cluster do not have direct iBGP peers among one another and the clients

exchange updates through the RR. If you configure peer groups, a potential withdrawal to the source of a route on the RR transmits to all clients inside the cluster. This transmission can cause problems.

The router subcommand [bgp client-to-client reflection](#) is enabled by default on the RR. If you turn off BGP client-to-client reflection on the RR and you make redundant BGP peering between the clients, you can safely use peer groups. Refer to [Limitations of Peer Groups](#) for more information.

RR and Conventional BGP Speakers

An AS can have BGP speakers that do not understand the concept of RRs. This document calls these routers conventional BGP speakers. The RR scheme allows such conventional BGP speakers to coexist. These routers can be either members of a client group or a nonclient group. The existence of these routers allows easy and gradual migration from the current iBGP model to the RR model. You can start to create clusters if you configure a single router as an RR and make other RRs and RR clients normal iBGP peers. Then, you can create more clusters gradually.



In this diagram, RTD, RTE, and RTF have the concept of route reflection. RTC, RTA, and RTB are "conventional" routers. You cannot configure these routers as RRs. You can do normal iBGP mesh between these routers and RTD. Later on, when you are ready to upgrade, you can make RTC an RR with clients RTA and RTB. Clients do not have to understand the route reflection

scheme; only the RRs require the upgrade.

Here is the configuration of RTD and RTC:

RTD#

```
router bgp 100
neighbor 6.6.6.6 remote-as 100
neighbor 6.6.6.6 route-reflector-client
neighbor 5.5.5.5 remote-as 100
neighbor 5.5.5.5 route-reflector-client
neighbor 3.3.3.3 remote-as 100
neighbor 2.2.2.2 remote-as 100
neighbor 1.1.1.1 remote-as 100
neighbor 13.13.13.13 remote-as 300
```

RTC#

```
router bgp 100
neighbor 4.4.4.4 remote-as 100
neighbor 2.2.2.2 remote-as 100
neighbor 1.1.1.1 remote-as 100
neighbor 14.14.14.14 remote-as 400
```

When you are ready to upgrade RTC and make RTC an RR, remove the iBGP full mesh and have RTA and RTB become clients of RTC.

[Avoid Loop of Routing Information](#)

So far, this document has mentioned two attributes that you can use to prevent potential information looping: **originator-id** and **cluster-list**.

Another means to control loops is to put more restrictions on the **set** clause of outbound route maps. The **set** clause for outbound route maps does not affect routes that reflect to iBGP peers.

You can also put more restrictions on **nexthop-self**, which is a per-neighbor configuration option. When you use **nexthop-self** on RRs, the clause only affects the next hop of eBGP learned routes because the next hop of reflected routes should not be changed.

[Route Flap Dampening](#)

Cisco IOS Software Release 11.0 introduced route dampening. Route dampening is a mechanism to minimize the instability that route flapping causes. Route dampening also reduces oscillation over the network. You define criteria to identify poorly behaved routes. A route that flaps gets a penalty of 1000 for each flap. As soon as the cumulative penalty reaches a predefined "suppress limit", suppression of the route advertisement occurs. The penalty decays exponentially based on a preconfigured "half-life time". Once the penalty decreases below a predefined "reuse limit", unsuppression of the route advertisement occurs.

Route dampening does not apply to routes that are external to an AS and learned via iBGP. In this way, route dampening avoids a higher penalty for the iBGP peers for routes external to the AS.

The penalty decays at a granularity of 5 seconds. Unsuppression of the routes is at a granularity of

10 seconds. The router keeps the dampening information until the penalty becomes less than half of the "reuse limit". At that point, the router purges the information.

Initially, dampening is off by default. If there is a need, this feature may be given default enablement in the future. These commands control route dampening:

bgp dampening—Turns on dampening.

no bgp dampening—Turns off dampening.

bgp dampening half-life-time —Changes the half-life time.

A command that sets all parameters at the same time is:

bgp dampening half-life-time reuse suppress maximum-suppress-time

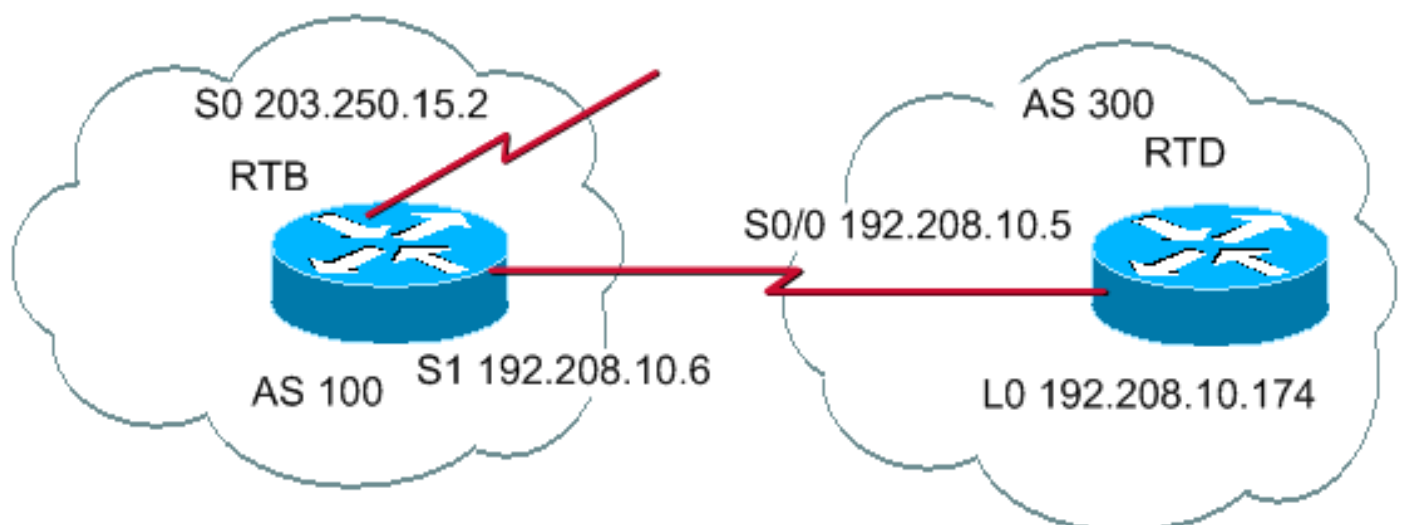
This list details the syntax:

half-life-time —The range is 1–45 minutes, and the current default is 15 minutes.

reuse-value —The range is 1–20,000, and the default is 750.

suppress-value —The range is 1–20,000, and the default is 2000.

max-suppress-time —This is the maximum duration for the suppression of a route. The range is 1–255 minutes, and the default is 4 times the half-life time.



```
RTB#  
hostname RTB
```

```

interface Serial0
 ip address 203.250.15.2 255.255.255.252

interface Serial1
 ip address 192.208.10.6 255.255.255.252

router bgp 100
 bgp dampening
 network 203.250.15.0
 neighbor 192.208.10.5 remote-as 300

```

```

RTD#
hostname RTD

interface Loopback0
 ip address 192.208.10.174 255.255.255.192

interface Serial0/0
 ip address 192.208.10.5 255.255.255.252

router bgp 300
 network 192.208.10.0
 neighbor 192.208.10.6 remote-as 100

```

The configuration of RTB is for route dampening with default parameters. If you assume that the eBGP link to RTD is stable, the RTB BGP table looks like this:

```

RTB# show ip bgp BGP table version is 24, local router ID is 203.250.15.2 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *> 192.208.10.0 192.208.10.5 0 0 300 i
*> 203.250.15.0 0.0.0.0 0 32768 i

```

In order to simulate a route flap, issue the **clear ip bgp 192.208.10.6** command on RTD. The RTB BGP table looks like this:

```

RTB# show ip bgp BGP table version is 24, local router ID is 203.250.15.2 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path h 192.208.10.0 192.208.10.5 0 0 300 i *>
203.250.15.0 0.0.0.0 0 32768 i

```

The BGP entry for 192.208.10.0 is in a `history` state. This placement means that you do not have a best path to the route, but information about the route flapping still exists.

```

RTB# show ip bgp 192.208.10.0 BGP routing table entry for 192.208.10.0 255.255.255.0, version 25
Paths: (1 available, no best path) 300 (history entry) 192.208.10.5 from 192.208.10.5
(192.208.10.174) Origin IGP, metric 0, external Dampinfo: penalty 910, flapped 1 times in
0:02:03

```

The route has received a penalty for flapping, but the penalty is still below the "suppress limit". The default is 2000. Route suppression has not yet occurred. If the route flaps a few more times, you see:

```

RTB# show ip bgp BGP table version is 32, local router ID is 203.250.15.2 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *d 192.208.10.0 192.208.10.5 0 0 300 i
*> 203.250.15.0 0.0.0.0 0 32768 i
RTB# show ip bgp 192.208.10.0 BGP routing table entry for

```


192.208.10.0 255.255.255.0, version 32 Paths: (1 available, no best path) 300, (suppressed due to dampening) 192.208.10.5 from 192.208.10.5 (192.208.10.174) Origin IGP, metric 0, valid, external Dampinfo: penalty 2615, flapped 3 times in 0:05:18 , reuse in 0:27:00

The route has been dampened, or suppressed. The route is reused when the penalty reaches the "reuse value". In this case, the reuse value is the default, 750. The dampening information is purged when the penalty becomes less than half of the reuse limit. In this case, the purge occurs when the penalty becomes 375 ($750/2=375$). These commands show and clear flap statistics information:

show ip bgp flap-statistics—Displays flap statistics for all the paths.

show ip bgp flap-statistics regexp *regular-expression* —Displays flap statistics for all paths that match the regular expression.

show ip bgp flap-statistics filter-list *list* —Displays flap statistics for all paths that pass the filter.

show ip bgp flap-statistics *A.B.C.D m.m.m.m* —Displays flap statistics for a single entry.

show ip bgp flap-statistics *A.B.C.D m.m.m.m longer-prefix*—Displays flap statistics for more specific entries.

show ip bgp neighbor [dampened-routes] | [flap-statistics] —Displays flap statistics for all paths from a neighbor.

clear ip bgp flap-statistics —Clears flap statistics for all routes.

clear ip bgp flap-statistics regexp *regular-expression* —Clears flap statistics for all the paths that match the regular expression.

clear ip bgp flap-statistics filter-list *list* —Clears flap statistics for all the paths that pass the filter.

clear ip bgp flap-statistics *A.B.C.D m.m.m.m* —Clears flap statistics for a single entry.

clear ip bgp *A.B.C.D flap-statistics*—Clears flap statistics for all paths from a neighbor.

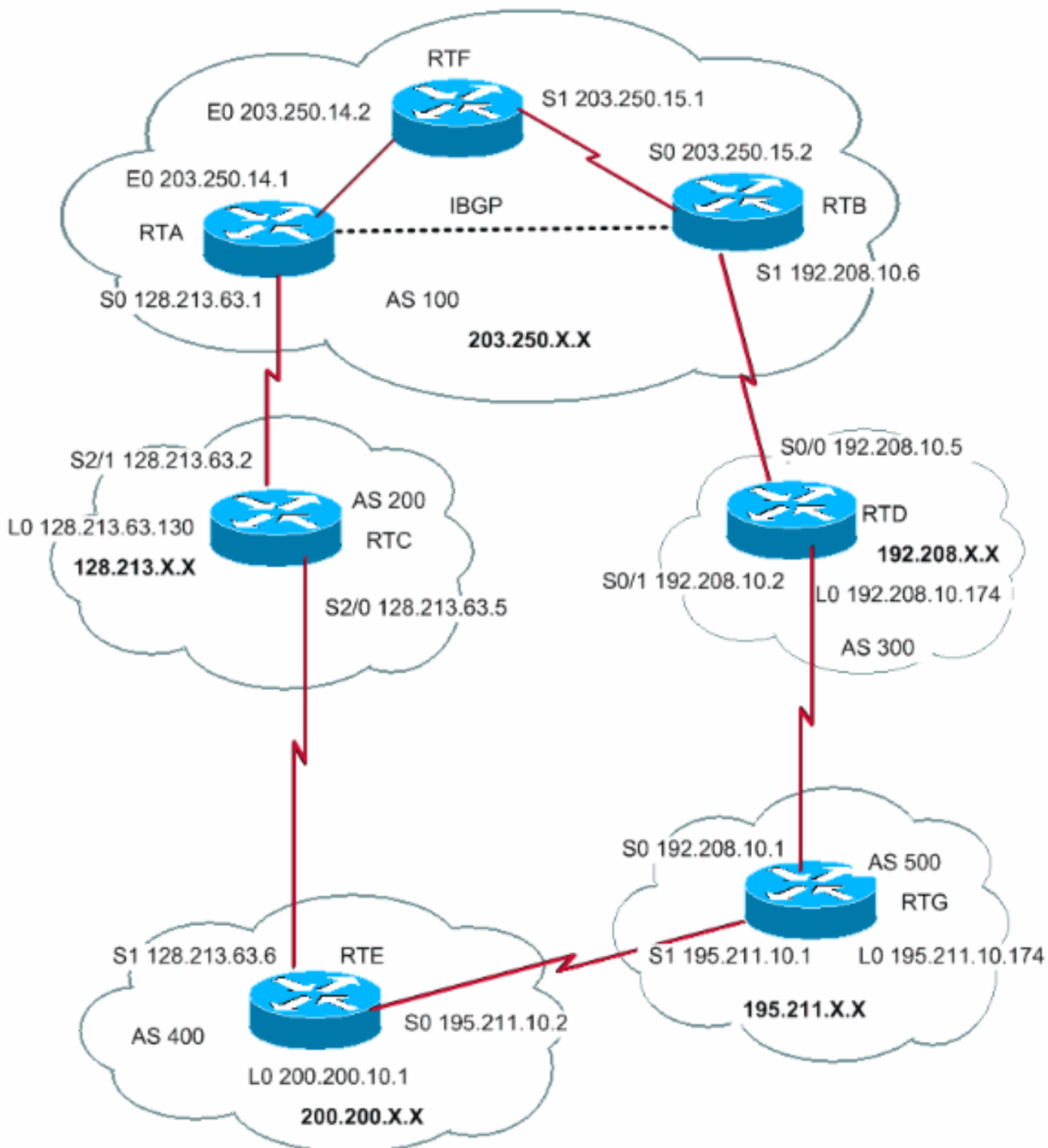
[How BGP Selects a Path](#)

Now that you are familiar with the BGP attributes and terminology, refer to [BGP Best Path Selection Algorithm](#).

[BGP Case Studies 5](#)

Practical Design Example

This section contains a design example that shows the configuration and routing tables as the tables actually appear on Cisco routers.



This section shows how to build this configuration step by step and what can go wrong along the way. Whenever you have an AS that connects to two ISPs via eBGP, always run iBGP within your AS in order to have better control of your routes. In this example, iBGP runs inside AS100 between RTA and RTB, and OSPF runs as an IGP. Assume that you connect to two ISPs, AS200 and AS300. This is the first run of the configurations for all the routers:

Note: These configurations are not the final configurations.

```
RTA#
hostname RTA

ip subnet-zero

interface Loopback0
 ip address 203.250.13.41 255.255.255.0

interface Ethernet0
 ip address 203.250.14.1 255.255.255.0

interface Serial0
 ip address 128.213.63.1 255.255.255.252

router ospf 10
 network 203.250.0.0 0.0.255.255 area 0

router bgp 100
 network 203.250.13.0
 network 203.250.14.0
 neighbor 128.213.63.2 remote-as 200
 neighbor 203.250.15.2 remote-as 100
 neighbor 203.250.15.2 update-source Loopback0
```

```
RTF#
hostname RTF

ip subnet-zero

interface Ethernet0
 ip address 203.250.14.2 255.255.255.0

interface Serial1
 ip address 203.250.15.1 255.255.255.252

router ospf 10
 network 203.250.0.0 0.0.255.255 area 0
```

```
RTB#
hostname RTB

ip subnet-zero

interface Serial0
 ip address 203.250.15.2 255.255.255.252

interface Serial1
 ip address 192.208.10.6 255.255.255.252

router ospf 10
 network 203.250.0.0 0.0.255.255 area 0
```

```
router bgp 100
network 203.250.15.0
  neighbor 192.208.10.5 remote-as 300
  neighbor 203.250.13.41 remote-as 100

RTC#
hostname RTC

ip subnet-zero

interface Loopback0
  ip address 128.213.63.130 255.255.255.192

interface Serial2/0
  ip address 128.213.63.5 255.255.255.252
!
interface Serial2/1
  ip address 128.213.63.2 255.255.255.252

router bgp 200
network 128.213.0.0
  neighbor 128.213.63.1 remote-as 100
  neighbor 128.213.63.6 remote-as 400

RTD#
hostname RTD

ip subnet-zero

interface Loopback0
ip address 192.208.10.174 255.255.255.192

interface Serial0/0
  ip address 192.208.10.5 255.255.255.252
!
interface Serial0/1
  ip address 192.208.10.2 255.255.255.252

router bgp 300
network 192.208.10.0
  neighbor 192.208.10.1 remote-as 500
  neighbor 192.208.10.6 remote-as 100

RTE#
hostname RTE

ip subnet-zero

interface Loopback0
ip address 200.200.10.1 255.255.255.0

interface Serial0
```

```
ip address 195.211.10.2 255.255.255.252
```

```
interface Serial1
```

```
ip address 128.213.63.6 255.255.255.252
```

```
clockrate 1000000
```

```
router bgp 400
```

```
network 200.200.10.0
```

```
neighbor 128.213.63.5 remote-as 200
```

```
neighbor 195.211.10.1 remote-as 500
```

```
RTG#
```

```
hostname RTG
```

```
ip subnet-zero
```

```
interface Loopback0
```

```
ip address 195.211.10.174 255.255.255.192
```

```
interface Serial0
```

```
ip address 192.208.10.1 255.255.255.252
```

```
interface Serial1
```

```
ip address 195.211.10.1 255.255.255.252
```

```
router bgp 500
```

```
network 195.211.10.0
```

```
neighbor 192.208.10.2 remote-as 300
```

```
neighbor 195.211.10.2 remote-as 400
```

Always use the **network** command or redistribute static entries into BGP to advertise networks. This method is better than a redistribution of IGP into BGP. This example uses the **network** command to inject networks into BGP.

Here, you start with the s1 interface on RTB shutdown, as if the link between RTB and RTD does not exist. This is the RTB BGP table:

```
RTB# show ip bgp BGP table version is 4, local router ID is 203.250.15.2 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *i128.213.0.0 128.213.63.2 0 100 0 200 i
*i192.208.10.0 128.213.63.2 100 0 200 400 500 300 i *i195.211.10.0 128.213.63.2 100 0 200 400
500 i *i200.200.10.0 128.213.63.2 100 0 200 400 i *>i203.250.13.0 203.250.13.41 0 100 0 i
*>i203.250.14.0 203.250.13.41 0 100 0 i *>203.250.15.0 0.0.0.0 0 32768 i
```

In this table, these notations appear:

An **i** at the beginning—Indicates that the entry was learned via an iBGP peer.

An **i** at the end—Indicates that the origin of the path information is IGP.

Path information—This information is intuitive. For example, network 128.213.0.0 is learned via path 200 with a next hop of 128.213.63.2.

Note: Any locally generated entry, such as 203.250.15.0, has a next hop 0.0.0.0.

An > symbol—Indicates that BGP has chosen the best route. BGP uses the decision steps that the document [BGP Best Path Selection Algorithm](#) outlines. BGP picks one best path to reach a destination, installs the path in the IP routing table, and advertises the path to other BGP peers.

Note: Notice the `Next Hop` attribute. RTB knows about 128.213.0.0 via a next hop of 128.213.63.2, which is the eBGP next hop carried into iBGP.

Look at the IP routing table:

```
RTB# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP D -
EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2 - OSPF
external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
default Gateway of last resort is not set 203.250.13.0 255.255.255.255 is subnetted, 1 subnets O
203.250.13.41 [110/75] via 203.250.15.1, 02:50:45, Serial0 203.250.15.0 255.255.255.252 is
subnetted, 1 subnets C 203.250.15.0 is directly connected, Serial0 O 203.250.14.0 [110/74] via
203.250.15.1, 02:50:46, Serial0
```

Apparently, none of the BGP entries has reached the routing table. Two problems exist here.

The first problem is that the next hop for these entries, 128.213.63.2, is unreachable. There is no way to reach that next hop via this IGP, which is OSPF. RTB has not learned about 128.213.63.0 via OSPF. You can run OSPF on the RTA s0 interface and make it passive; in this way, RTB knows how to reach the next hop 128.213.63.2. This RTA configuration appears here:

```
RTA#
hostname RTA

ip subnet-zero

interface Loopback0
 ip address 203.250.13.41 255.255.255.0

interface Ethernet0
 ip address 203.250.14.1 255.255.255.0

interface Serial0
 ip address 128.213.63.1 255.255.255.252

router ospf 10
 passive-interface Serial0
 network 203.250.0.0 0.0.255.255 area 0
 network 128.213.0.0 0.0.255.255 area 0

router bgp 100
 network 203.250.0.0 mask 255.255.0.0
 neighbor 128.213.63.2 remote-as 200
 neighbor 203.250.15.2 remote-as 100
 neighbor 203.250.15.2 update-source Loopback0
```

Note: You can issue the `bgp nexthopself` command between RTA and RTB in order to change

the next hop.

The new BGP table on RTB looks like this:

```
RTB# show ip bgp BGP table version is 10, local router ID is 203.250.15.2 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *>i128.213.0.0 128.213.63.2 0 100 0 200
i *>i192.208.10.0 128.213.63.2 100 0 200 400 500 300 i *>i195.211.10.0 128.213.63.2 100 0 200
400 500 i *>i200.200.10.0 128.213.63.2 100 0 200 400 i *>i203.250.13.0 203.250.13.41 0 100 0 i
*>i203.250.14.0 203.250.13.41 0 100 0 i *> 203.250.15.0 0.0.0.0 0 32768 i
```

Note: All the entries have >, which means that BGP can reach the next hop.

Look at the routing table:

```
RTB# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP D -
EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2 - OSPF
external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
default Gateway of last resort is not set 203.250.13.0 255.255.255.255 is subnetted, 1 subnets O
203.250.13.41 [110/75] via 203.250.15.1, 00:04:46, Serial0 203.250.15.0 255.255.255.252 is
subnetted, 1 subnets C 203.250.15.0 is directly connected, Serial0 O 203.250.14.0 [110/74] via
203.250.15.1, 00:04:46, Serial0 128.213.0.0 255.255.255.252 is subnetted, 1 subnets O
128.213.63.0 [110/138] via 203.250.15.1, 00:04:47, Serial0
```

The second problem is that you still do not see the BGP entries in the routing table. The only difference is that 128.213.63.0 is now reachable via OSPF. This problem is a synchronization issue. BGP does not put these entries in the routing table and does not send the entries in BGP updates because of a lack of synchronization with the IGP.

Note: RTF has no notion of networks 192.208.10.0 and 195.211.10.0 because you have not redistributed BGP into OSPF yet.

In this scenario, if you turn synchronization off, the entries appear in the routing table. But connectivity is still broken.

If you turn off synchronization on RTB, this is what happens:

```
RTB# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP D -
EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2 - OSPF
external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
default Gateway of last resort is not set B 200.200.10.0 [200/0] via 128.213.63.2, 00:01:07 B
195.211.10.0 [200/0] via 128.213.63.2, 00:01:07 B 192.208.10.0 [200/0] via 128.213.63.2,
00:01:07 203.250.13.0 is variably subnetted, 2 subnets, 2 masks O 203.250.13.41 255.255.255.255
[110/75] via 203.250.15.1, 00:12:37, Serial0 B 203.250.13.0 255.255.255.0 [200/0] via
203.250.13.41, 00:01:08 203.250.15.0 255.255.255.252 is subnetted, 1 subnets C 203.250.15.0 is
directly connected, Serial0 O 203.250.14.0 [110/74] via 203.250.15.1, 00:12:37, Serial0
128.213.0.0 is variably subnetted, 2 subnets, 2 masks B 128.213.0.0 255.255.0.0 [200/0] via
128.213.63.2, 00:01:08 O 128.213.63.0 255.255.255.252 [110/138] via 203.250.15.1, 00:12:37,
Serial0
```

The routing table looks fine, but there is no way to reach those networks. RTF in the middle does not know how to reach the networks:

```
RTF# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP D -
EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2 - OSPF
external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
```

default Gateway of last resort is not set 203.250.13.0 255.255.255.255 is subnetted, 1 subnets O 203.250.13.41 [110/11] via 203.250.14.1, 00:14:15, Ethernet0 203.250.15.0 255.255.255.252 is subnetted, 1 subnets C 203.250.15.0 is directly connected, Serial1 C 203.250.14.0 is directly connected, Ethernet0 128.213.0.0 255.255.255.252 is subnetted, 1 subnets O 128.213.63.0 [110/74] via 203.250.14.1, 00:14:15, Ethernet0

When you turn off synchronization in this situation, the problem still exists. But you need synchronization later for other issues. Redistribute BGP into OSPF on RTA, with a metric of 2000:

```
RTA#
hostname RTA

ip subnet-zero

interface Loopback0
 ip address 203.250.13.41 255.255.255.0

interface Ethernet0
 ip address 203.250.14.1 255.255.255.0

interface Serial0
 ip address 128.213.63.1 255.255.255.252

router ospf 10
 redistribute bgp 100 metric 2000 subnets
 passive-interface Serial0
 network 203.250.0.0 0.0.255.255 area 0
 network 128.213.0.0 0.0.255.255 area 0

router bgp 100
 network 203.250.0.0 mask 255.255.0.0
 neighbor 128.213.63.2 remote-as 200
 neighbor 203.250.15.2 remote-as 100
 neighbor 203.250.15.2 update-source Loopback0
```

The routing table looks like this:

```
RTB# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP D -
EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2 - OSPF
external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
default Gateway of last resort is not set O E2 200.200.10.0 [110/2000] via 203.250.15.1,
00:00:14, Serial0 O E2 195.211.10.0 [110/2000] via 203.250.15.1, 00:00:14, Serial0 O E2
192.208.10.0 [110/2000] via 203.250.15.1, 00:00:14, Serial0 203.250.13.0 is variably subnetted,
2 subnets, 2 masks O 203.250.13.41 255.255.255.255 [110/75] via 203.250.15.1, 00:00:15, Serial0
O E2 203.250.13.0 255.255.255.0 [110/2000] via 203.250.15.1, 00:00:15, Serial0 203.250.15.0
255.255.255.252 is subnetted, 2 subnets C 203.250.15.8 is directly connected, Loopback1 C
203.250.15.0 is directly connected, Serial0 O 203.250.14.0 [110/74] via 203.250.15.1, 00:00:15,
Serial0 128.213.0.0 is variably subnetted, 2 subnets, 2 masks O E2 128.213.0.0 255.255.0.0
[110/2000] via 203.250.15.1, 00:00:15, Serial0 O 128.213.63.0 255.255.255.252 [110/138] via
203.250.15.1, 00:00:16, Serial0
```

The BGP entries have disappeared because OSPF has a better distance than iBGP. The OSPF distance is 110, while the iBGP distance is 200.

Turn off synchronization on RTA so that RTA can advertise 203.250.15.0. This action is necessary

because RTA does not synchronize with OSPF because of the difference in masks. Keep synchronization off on RTB so that RTB can advertise 203.250.13.0. This action is necessary on RTB for the same reason.

Now, bring up the RTB s1 interface to see what the routes look like. Also, enable OSPF on serial 1 of RTB to make it passive. This step allows RTA to know about the next hop 192.208.10.5 via IGP. If you do not take this step, routing loops occur because, in order to reach next hop 192.208.10.5, you need to go the other way via eBGP. These are the new configurations of RTA and RTB:

```
RTA#
hostname RTA

ip subnet-zero

interface Loopback0
 ip address 203.250.13.41 255.255.255.0

interface Ethernet0
 ip address 203.250.14.1 255.255.255.0

interface Serial0
 ip address 128.213.63.1 255.255.255.252

router ospf 10
 redistribute bgp 100 metric 2000 subnets
 passive-interface Serial0
 network 203.250.0.0 0.0.255.255 area 0
 network 128.213.0.0 0.0.255.255 area 0

router bgp 100
 no synchronization
 network 203.250.13.0
 network 203.250.14.0
 neighbor 128.213.63.2 remote-as 200
 neighbor 203.250.15.2 remote-as 100
 neighbor 203.250.15.2 update-source Loopback0
```

```
RTB#
hostname RTB

ip subnet-zero

interface Serial0
 ip address 203.250.15.2 255.255.255.252

interface Serial1
 ip address 192.208.10.6 255.255.255.252

router ospf 10
 redistribute bgp 100 metric 1000 subnets
 passive-interface Serial1
 network 203.250.0.0 0.0.255.255 area 0
```

```
network 192.208.0.0 0.0.255.255 area 0
```

```
router bgp 100
no synchronization
network 203.250.15.0
neighbor 192.208.10.5 remote-as 300
neighbor 203.250.13.41 remote-as 100
```

The BGP tables look like this:

```
RTA# show ip bgp BGP table version is 117, local router ID is 203.250.13.41 Status codes: s
suppressed, d damped, h history, * valid, > best, i -internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *> 128.213.0.0 128.213.63.2 0 0 200 i
*>i192.208.10.0 192.208.10.5 0 100 0 300 i *>i195.211.10.0 192.208.10.5 100 0 300 500 i *
128.213.63.2 0 200 400 500 i *> 200.200.10.0 128.213.63.2 0 200 400 i *> 203.250.13.0 0.0.0.0 0
32768 i *> 203.250.14.0 0.0.0.0 0 32768 i *>i203.250.15.0 203.250.15.2 0 100 0 i
RTB# show ip
bgp BGP table version is 12, local router ID is 203.250.15.10 Status codes: s suppressed, d
damped, h history, * valid, > best, i -internal Origin codes: i - IGP, e - EGP, ? - incomplete
Network Next Hop Metric LocPrf Weight Path *>i128.213.0.0 128.213.63.2 0 100 0 200 i *
192.208.10.5 0 300 500 400 200 i *> 192.208.10.0 192.208.10.5 0 0 300 i *> 195.211.10.0
192.208.10.5 0 300 500 i *>i200.200.10.0 128.213.63.2 100 0 200 400 i * 192.208.10.5 0 300 500
400 i *>i203.250.13.0 203.250.13.41 0 100 0 i *>i203.250.14.0 203.250.13.41 0 100 0 i *>
203.250.15.0 0.0.0.0 0 32768 i
```

There are multiple ways to design your network to talk to the two different ISPs, AS200 and AS300. One way is to have a primary ISP and a backup ISP. You can learn partial routes from one of the ISPs and default routes to both ISPs. In this example, you receive partial routes from AS200 and only local routes from AS300. Both RTA and RTB generate default routes into OSPF, with RTB as the preference because of the lower metric. In this way, you can balance outgoing traffic between the two ISPs.

Potential asymmetry can occur if traffic that leaves RTA comes back via RTB. This situation can occur if you use the same pool of IP addresses, the same major net, when you talk to the two ISPs. Because of aggregation, your whole AS can look like one whole entity to the outside world. Entry points to your network can occur via RTA or RTB. You can discover that all incoming traffic to your AS arrives via one single point, even though you have multiple points to the Internet. In the example, you have two different major nets when you talk to the two ISPs.

Another potential reason for asymmetry is the different advertised path length to reach your AS. Perhaps one service provider is closer to a certain destination than another. In the example, traffic from AS400 that has your network as the destination always comes in via RTA because of the shorter path. You can try to effect that decision. You can use the **set as-path prepend** command in order to prepend path numbers to your updates and make the path length look longer. But, with attributes such as local preference, metric, or weight, AS400 can have set the exit point to be AS200. In this case, there is nothing that you can do.

This configuration is the final configuration for all the routers:

```
RTA#
hostname RTA

ip subnet-zero

interface Loopback0
ip address 203.250.13.41 255.255.255.0
```

```
interface Ethernet0
 ip address 203.250.14.1 255.255.255.0

interface Serial0
 ip address 128.213.63.1 255.255.255.252

router ospf 10
 redistribute bgp 100 metric 2000 subnets
 passive-interface Serial0
 network 203.250.0.0 0.0.255.255 area 0
 network 128.213.0.0 0.0.255.255 area 0
 default-information originate metric 2000

router bgp 100
 no synchronization
 network 203.250.13.0
 network 203.250.14.0
 neighbor 128.213.63.2 remote-as 200
 neighbor 128.213.63.2 route-map setlocalpref in
 neighbor 203.250.15.2 remote-as 100
 neighbor 203.250.15.2 update-source Loopback0

ip classless
ip default-network 200.200.0.0
```

```
route-map setlocalpref permit 10
 set local-preference 200
```

On RTA, the local preference for routes that come from AS200 is set to 200. Also, network 200.200.0.0 is the choice for the candidate default. The **ip default-network** command enables you to choose the default.

Also in this example, use of the [default-information originate](#) command with OSPF injects the default route inside the OSPF domain. This example also uses this command with Intermediate System-to-Intermediate System Protocol (IS-IS Protocol) and BGP. For RIP, there is an automatic redistribution into RIP of 0.0.0.0, without additional configuration. For IGRP and EIGRP, injection of the default information into the IGP domain occurs after redistribution of BGP into IGRP and EIGRP. Also, with IGRP and EIGRP, you can redistribute a static route to 0.0.0.0 into the IGP domain.

```
RTF#
hostname RTF

ip subnet-zero

interface Ethernet0
 ip address 203.250.14.2 255.255.255.0

interface Serial1
 ip address 203.250.15.1 255.255.255.252

router ospf 10
```

```

network 203.250.0.0 0.0.255.255 area 0

ip classless

RTB#
hostname RTB

ip subnet-zero

interface Loopback1
 ip address 203.250.15.10 255.255.255.252

interface Serial0
 ip address 203.250.15.2 255.255.255.252
!
interface Serial1
 ip address 192.208.10.6 255.255.255.252

router ospf 10
 redistribute bgp 100 metric 1000 subnets
 passive-interface Serial1
 network 203.250.0.0 0.0.255.255 area 0
 network 192.208.10.6 0.0.0.0 area 0
 default-information originate metric 1000
!
router bgp 100
 no synchronization
 network 203.250.15.0
 neighbor 192.208.10.5 remote-as 300
 neighbor 192.208.10.5 route-map localonly in
 neighbor 203.250.13.41 remote-as 100
!
ip classless
ip default-network 192.208.10.0
ip as-path access-list 1 permit ^300$

```

```

route-map localonly permit 10
 match as-path 1
 set local-preference 300

```

For RTB, the local preference for updates that come from AS300 is set to 300. This value is higher than the local preference value of iBGP updates that come from RTA. In this way, AS100 picks RTB for the local routes of AS300. Any other routes on RTB, if other routes exist, transmit internally with a local preference of 100. This value is lower than the local preference of 200, which comes from RTA. So RTA is the preference.

Note: You only advertised the AS300 local routes. Any path information that does not match `^300$` drops. If you want to advertise the local routes and the neighbor routes, which are the customers of the ISP, use `^300_[0-9]*`.

Here is the output of the regular expression that indicates the AS300 local routes:

```

RTB# show ip bgp regexp ^300$ BGP table version is 14, local router ID is 203.250.15.10 Status

```

codes: s suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ? - incomplete Network Next Hop Metric LocPrf Weight Path *> 192.208.10.0 192.208.10.5 0 300 0 300 RTC# hostname RTC ip subnet-zero interface Loopback0 ip address 128.213.63.130 255.255.255.192 interface Serial2/0 ip address 128.213.63.5 255.255.255.252 ! interface Serial2/1 ip address 128.213.63.2 255.255.255.252 router bgp 200 network 128.213.0.0 neighbor 128.213.63.1 remote-as 100 neighbor 128.213.63.1 distribute-list 1 out neighbor 128.213.63.6 remote-as 400 ip classless access-list 1 deny 195.211.0.0 0.0.255.255 access-list 1 permit any

On RTC, you aggregate 128.213.0.0/16 and indicate the specific routes for injection into AS100. If the ISP refuses to do this task, you must filter on the incoming end of AS100.

RTD#

hostname RTD

ip subnet-zero

interface Loopback0

ip address 192.208.10.174 255.255.255.192

!

interface Serial0/0

ip address 192.208.10.5 255.255.255.252

!

interface Serial0/1

ip address 192.208.10.2 255.255.255.252

router bgp 300

network 192.208.10.0

neighbor 192.208.10.1 remote-as 500

neighbor 192.208.10.6 remote-as 100

RTG#

hostname RTG

ip subnet-zero

interface Loopback0

ip address 195.211.10.174 255.255.255.192

interface Serial0

ip address 192.208.10.1 255.255.255.252

interface Serial1

ip address 195.211.10.1 255.255.255.252

router bgp 500

network 195.211.10.0

aggregate-address 195.211.0.0 255.255.0.0 summary-only

neighbor 192.208.10.2 remote-as 300

neighbor 192.208.10.2 send-community

neighbor 192.208.10.2 route-map setcommunity out

neighbor 195.211.10.2 remote-as 400

!

ip classless

```

access-list 1 permit 195.211.0.0 0.0.255.255
access-list 2 permit any
route-map setcommunity permit 20
  match ip address 2
!
route-map setcommunity permit 10
  match ip address 1
  set community no-export

```

A demonstration of the use of community filtering is on RTG. You add a **no-export** community to 195.211.0.0 updates toward RTD. In this way, RTD does not export that route to RTB. However, in this case, RTB does not accept these routes anyway.

```

RTE#
hostname RTE

ip subnet-zero

interface Loopback0
 ip address 200.200.10.1 255.255.255.0

interface Serial0
 ip address 195.211.10.2 255.255.255.252

interface Serial1
 ip address 128.213.63.6 255.255.255.252

router bgp 400
 network 200.200.10.0
 aggregate-address 200.200.0.0 255.255.0.0 summary-only
 neighbor 128.213.63.5 remote-as 200
 neighbor 195.211.10.1 remote-as 500

```

```
ip classless
```

RTE aggregates 200.200.0.0/16. Here are the final BGP and routing tables for RTA, RTF, and RTB:

```

RTA# show ip bgp BGP table version is 21, local router ID is 203.250.13.41 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *> 128.213.0.0 128.213.63.2 0 200 0 200
i *>i192.208.10.0 192.208.10.5 0 300 0 300 i *> 200.200.0.0/16 128.213.63.2 200 0 200 400 i *>
203.250.13.0 0.0.0.0 0 32768 i *> 203.250.14.0 0.0.0.0 0 32768 i *>i203.250.15.0 203.250.15.2 0
100 0 i RTA# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B -
BGP D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF external type 1, E2
- OSPF external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate
default Gateway of last resort is 128.213.63.2 to network 200.200.0.0 192.208.10.0 is variably
subnetted, 2 subnets, 2 masks O E2 192.208.10.0 255.255.255.0 [110/1000] via 203.250.14.2,
00:41:25, Ethernet0 O 192.208.10.4 255.255.255.252 [110/138] via 203.250.14.2, 00:41:25,
Ethernet0 C 203.250.13.0 is directly connected, Loopback0 203.250.15.0 is variably subnetted, 3
subnets, 3 masks O 203.250.15.10 255.255.255.255 [110/75] via 203.250.14.2, 00:41:25, Ethernet0
O 203.250.15.0 255.255.255.252 [110/74] via 203.250.14.2, 00:41:25, Ethernet0 B 203.250.15.0
255.255.255.0 [200/0] via 203.250.15.2, 00:41:25 C 203.250.14.0 is directly connected, Ethernet0
128.213.0.0 is variably subnetted, 2 subnets, 2 masks B 128.213.0.0 255.255.0.0 [20/0] via

```

```

128.213.63.2, 00:41:26 C 128.213.63.0 255.255.255.252 is directly connected, Serial0 O*E2
0.0.0.0/0 [110/1000] via 203.250.14.2, Ethernet0/0 B* 200.200.0.0 255.255.0.0 [20/0] via
128.213.63.2, 00:02:38 RTF# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M
- mobile, B - BGP D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF
external type 1, E2 - OSPF external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS
level-2, * - candidate default Gateway of last resort is 203.250.15.2 to network 0.0.0.0
192.208.10.0 is variably subnetted, 2 subnets, 2 masks O E2 192.208.10.0 255.255.255.0
[110/1000] via 203.250.15.2, 00:48:50, Serial1 O 192.208.10.4 255.255.255.252 [110/128] via
203.250.15.2, 01:12:09, Serial1 203.250.13.0 is variably subnetted, 2 subnets, 2 masks O
203.250.13.41 255.255.255.255 [110/11] via 203.250.14.1, 01:12:09, Ethernet0 O E2 203.250.13.0
255.255.255.0 [110/2000] via 203.250.14.1, 01:12:09, Ethernet0 203.250.15.0 is variably
subnetted, 2 subnets, 2 masks O 203.250.15.10 255.255.255.255 [110/65] via 203.250.15.2,
01:12:09, Serial1 C 203.250.15.0 255.255.255.252 is directly connected, Serial1 C 203.250.14.0
is directly connected, Ethernet0 128.213.0.0 is variably subnetted, 2 subnets, 2 masks O E2
128.213.0.0 255.255.0.0 [110/2000] via 203.250.14.1, 00:45:01, Ethernet0 O 128.213.63.0
255.255.255.252 [110/74] via 203.250.14.1, 01:12:11, Ethernet0 O E2 200.200.0.0 255.255.0.0
[110/2000] via 203.250.14.1, 00:03:47, Ethernet0 O*E2 0.0.0.0 0.0.0.0 [110/1000] via
203.250.15.2, 00:03:33, Serial1

```

Note: The RTF routing table indicates that the way to reach networks local to AS300, such as 192.208.10.0, is through RTB. The way to reach other known networks, such as 200.200.0.0, is through RTA. The gateway of last resort is set to RTB. If something happens to the connection between RTB and RTD, the default that RTA advertises kicks in with a metric of 2000.

```

RTB# show ip bgp BGP table version is 14, local router ID is 203.250.15.10 Status codes: s
suppressed, d damped, h history, * valid, > best, i - internal Origin codes: i - IGP, e - EGP, ?
- incomplete Network Next Hop Metric LocPrf Weight Path *>i128.213.0.0 128.213.63.2 0 200 0 200
i *> 192.208.10.0 192.208.10.5 0 300 0 300 i *>i200.200.0.0/16 128.213.63.2 200 0 200 400 i
*>i203.250.13.0 203.250.13.41 0 100 0 i *>i203.250.14.0 203.250.13.41 0 100 0 i *> 203.250.15.0
0.0.0.0 0 32768 i RTB# show ip route Codes: C - connected, S - static, I - IGRP, R - RIP, M -
mobile, B - BGP D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area E1 - OSPF
external type 1, E2 - OSPF external type 2, E - EGP i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS
level-2, * - candidate default Gateway of last resort is 192.208.10.5 to network 192.208.10.0 *
192.208.10.0 is variably subnetted, 2 subnets, 2 masks B* 192.208.10.0 255.255.255.0 [20/0] via
192.208.10.5, 00:50:46 C 192.208.10.4 255.255.255.252 is directly connected, Serial1
203.250.13.0 is variably subnetted, 2 subnets, 2 masks O 203.250.13.41 255.255.255.255 [110/75]
via 203.250.15.1, 01:20:33, Serial0 O E2 203.250.13.0 255.255.255.0 [110/2000] via 203.250.15.1,
01:15:40, Serial0 203.250.15.0 255.255.255.252 is subnetted, 2 subnets C 203.250.15.8 is
directly connected, Loopback1 C 203.250.15.0 is directly connected, Serial0 O 203.250.14.0
[110/74] via 203.250.15.1, 01:20:33, Serial0 128.213.0.0 is variably subnetted, 2 subnets, 2
masks O E2 128.213.0.0 255.255.0.0 [110/2000] via 203.250.15.1, 00:46:55, Serial0 O 128.213.63.0
255.255.255.252 [110/138] via 203.250.15.1, 01:20:34, Serial0 O*E2 0.0.0.0/0 [110/2000] via
203.250.15.1, 00:08:33, Serial0 O E2 200.200.0.0 255.255.0.0 [110/2000] via 203.250.15.1,
00:05:42, Serial0

```

[Related Information](#)

- [BGP: Frequently Asked Questions](#)
- [Sample Configurations of BGP Across a PIX Firewall](#)
- [How to Use HSRP to Provide Redundancy in a Multihomed BGP Network](#)
- [Configuring Single Router Mode Redundancy and BGP on a Cat6000 MSFC](#)
- [Achieve Optimal Routing and Reduce BGP Memory Consumption](#)
- [Troubleshooting BGP](#)

- [Troubleshooting High CPU Caused by the BGP Scanner or BGP Router Process](#)
- [Load Sharing with BGP in Single and Multihomed Environments: Sample Configurations](#)
- [BGP Support Page](#)
- [Technical Support & Documentation - Cisco Systems](#)