- Border Gateway Protocol -

Border Gateway Protocol (BGP)

BGP is a standardized *exterior* gateway protocol (EGP), as opposed to RIP, OSPF, and EIGRP which are *interior* gateway protocols (IGP's). BGP Version 4 (**BGPv4)** is the current standard deployment.

BGP is considered a "Path Vector" routing protocol. BGP was not built to route *within* an Autonomous System (AS), but rather to route *between* AS's. BGP maintains a **separate routing table** based on shortest **AS Path** and various other attributes, as opposed to IGP metrics like distance or cost.

BGP is the routing protocol of choice on the Internet. Essentially, the Internet is a collection of interconnected Autonomous Systems.

BGP Autonomous Systems are assigned an Autonomous System Number (ASN), which is a 16-bit number ranging from **1 – 65535**. A specific subset of this range, **64512 – 65535**, has been reserved for private (or internal) use.

BGP utilizes **TCP** for reliable transfer of its packets, on **port 179**.

When to Use BGP

Contrary to popular opinion, BGP is not a necessity when multiple connections to the Internet are required. Fault tolerance or redundancy of outbound traffic can easily be handled by an IGP, such as OSPF or EIGRP.

BGP is also completely unnecessary if there is only one connection to an external AS (such as the Internet). There are over 100,000 routes on the Internet, and interior routers should not be needlessly burdened.

BGP should be used under the following circumstances:

- Multiple connections exist to external AS's (such as the Internet) via different providers.
- Multiple connections exist to external AS's through the same provider, but connect via a separate CO or routing policy.
- The existing routing equipment can handle the additional demands.

BGP's true benefit is in controlling how traffic *enters* the local AS, rather than how traffic *exits* it.

* * *

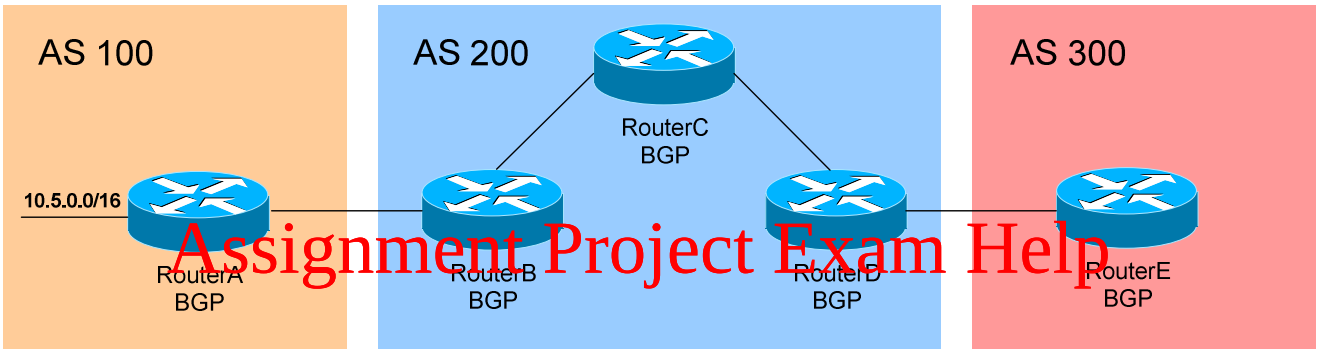
BGP Peers (Neighbors)

For BGP to function, BGP routers (called **speakers)** must form neighbor relationships (called **peers**).

There are two types of BGP neighbor relationships:

- **iBGP Peers** – BGP neighbors within the same autonomous system.
- **eBGP Peers** – BGP neighbors connecting separate autonomous systems.

Note: Do not confuse an *IGP*, such as OSPF, with *iBGP*!



In the above figure, RouterB and RouterC in AS 200 would form an **iBGP** peer relationship. RouterA in AS 100 and RouterB in AS 200 would form an **eBGP** peering.

Once BGP peers form a neighbor relationship, they share their full routing table. Afterwards, only changes to the routing table are forwarded to peers.

By default, BGP assumes that eBGP peers are a maximum of one hop away. This restriction can be bypassed using the *ebgp-multihop* option with the *neighbor* command (demonstrated later in this guide).

iBGP peers do not have a hop restriction, and are dependent on the underlying IGP of the AS to connect peers together. By default, all iBGP peers must be **fully meshed** within the Autonomous System.

A Cisco router running BGP can belong to **only one AS**. The IOS will only allow one BGP process to run on a router.

The Administrative Distance for routes learned outside the Autonomous System (eBGP routes) is **20**, while the AD for iBGP and locally-originated routes is **200**.

* * *

BGP Peers Messages

BGP forms its peer relationships through a series of **messages**. First, an **OPEN** message is sent between peers to initiate the session. The **OPEN** message contains several parameters:

- BGP Version – must be the same between BGP peers
- Local AS Number
- BGP Router ID

KEEPALIVE messages are sent periodically (every **60 seconds** by default) to ensure that the remote peer is still available. If a router does not receive a KEEPALIVE from a peer for a Hold-time period (by default, **180 seconds**), the router declares that peer dead.

UPDATE messages are used to exchange routes between peers.

Finally, **NOTIFICATION** messages are sent when there is a fatal error condition. If a NOTIFICATION message is sent, the BGP peer session is torn down and reset.

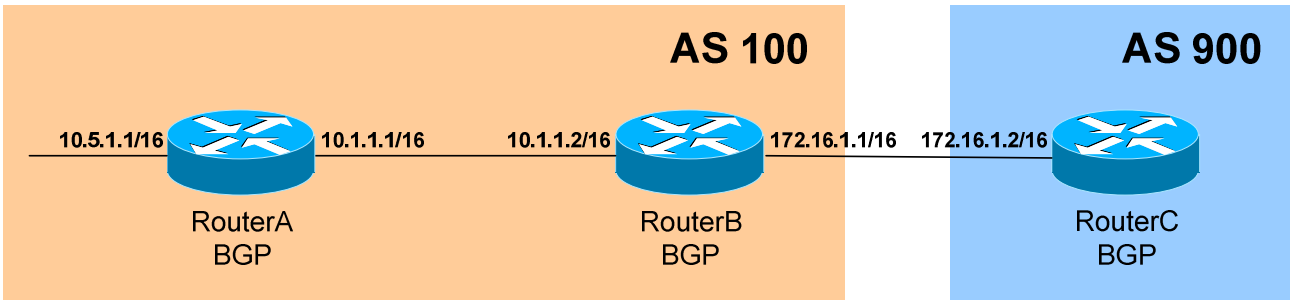
As a BGP peer session is forming, it will pass through several **states**. This process is known as the BGP **Finite State Machine** (FSM):

- **Idle** – the initial BGP state
- **Connect** - BGP waits for a TCP connection with the remote peer. If successful, an OPEN message is sent. If unsuccessful, the session is placed in an Active state.
- **Active** – BGP attempts to initiate a TCP connection with the remote peer. If successful, an OPEN message is sent. If unsuccessful, BGP will wait for a ConnectRetry timer to expire, and place the session back in a Connect State.
- **OpenSent** – BGP has both established the TCP connection *and* sent an OPEN Message, and is awaiting a reply OPEN Message. Once it receives a reply OPEN Message, the BGP peer will send a KEEPALIVE message.
- **OpenConfirm –** BGP listens for a reply KEEPALIVE message.
- **Established –** the BGP peer session is fully established. UPDATE messages containing routing information will now be sent.

If a peer session is stuck in an **Active** state, potential problems can include: no IP connectivity (no route to host), an incorrect *neighbor* statement, or an access-list filtering TCP port 179.

* * *

Configuring BGP Neighbors



The first step in configuring BGP is to enable the BGP process, and specify the router's Autonomous System (AS):

```
RouterB(config)# router bgp 100
```

RouterB is now a member of AS *100*. Next, neighbor relationships must be established. To configure a neighbor relationship with a router in the *same* AS (iBGP Peer):


```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 10.1.1.1 remote-as 100
```


To configure a neighbor relationship with a router in a *separate* AS (eBGP Peer):


```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 remote-as 900
```

Notice that the syntax is the same, and that the *remote-as* argument is always used, regardless if the peering is iBGP or eBGP.

For stability purposes, the *source* interface used to generate updates to a particular neighbor can be specified:

```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 update-source lo0
```

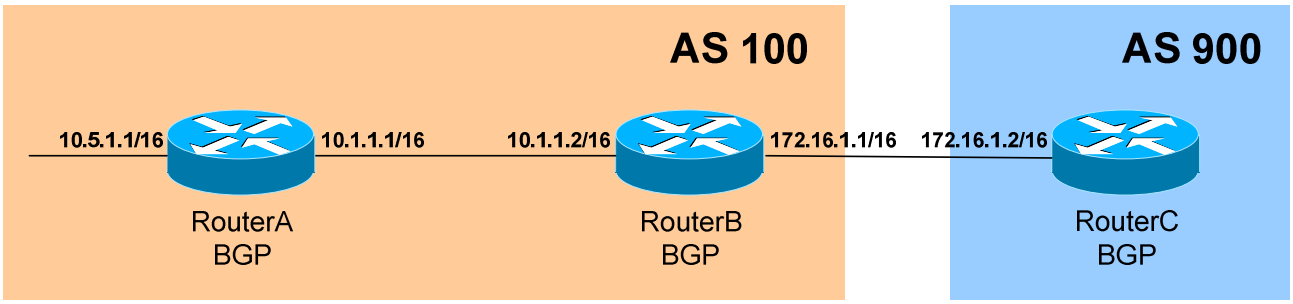
RouterC must then point to RouterB's loopback (assume the address is 1.1.1.1/24) in its neighbor statement:

```
RouterC(config)# router bgp 900
RouterC(config-router)# neighbor 1.1.1.1 remote-as 100
```

RouterC *must* have a route to RouterB's loopback in its routing table.

* * *

Configuring BGP Neighbors (continued)



Remember though: by default, BGP assumes that external peers are exactly one hop away. Using the loopback as a source interface puts RouterB two hops away from RouterC. Thus, the *ebgp-multihop* feature must be enabled:

```
RouterC(config)# router bgp 900
RouterC(config-router)# neighbor 1.1.1.1 ebgp-multihop 2
```

The *2* indicates the number of hops to the eBGP peer. If left blank, the default is 255.

To authenticate updates between two BGP peers:


```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 password CISCO
```


Configuring BGP Timers

To globally adjust the Keepalive and Hold-time timers for all neighbors:

```
RouterB(config)# router bgp 100
RouterB(config-router)# timers bgp 30 90
```

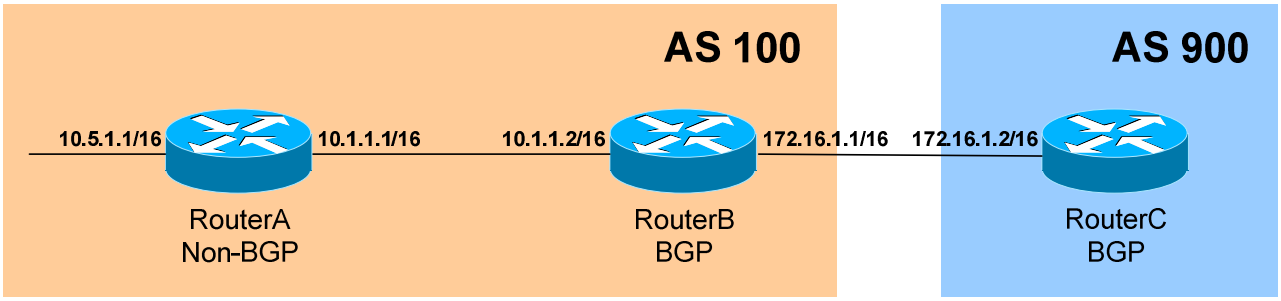
The above command sets the Keepalive timer to *30* seconds, and the Hold-time timer to *90* seconds. If the configured Hold-time timers between two peers are different, the peer session **will still** be established, and the **smallest** timer value will be used.

To adjust the timers for a *specific* neighbor (which overrides the global timer configuration):

```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 timers 30 90
```

* * *

Viewing BGP Neighbors



To view the status of *all* BGP neighbors:

RouterB# *show ip bgp neighbors*

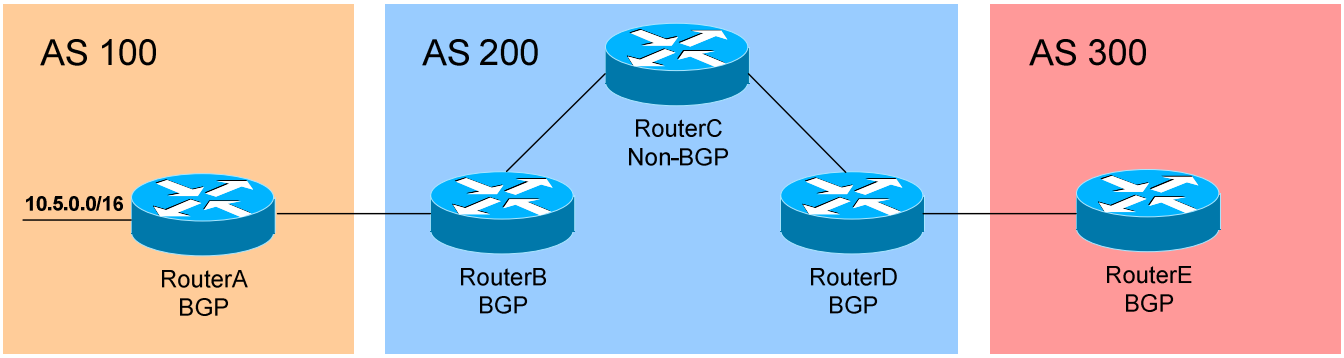
```
BGP neighbor is 172.16.1.2, remote AS 900, external link
 Index 1, Offset 0, Mask 0x2
  Inbound soft reconfiguration allowed
  BGP version 4, remote router ID 172.16.1.2
  BGP state = Established, table version = 27, up for 00:03:45
  Last read 00:00:19, hold time is 180, keepalive interval is 60
seconds
  Minimum time between advertisement runs is 30 seconds
  Received 25 messages, 0 notifications, 0 in queue
  Sent 20 messages, 0 notifications, 0 in queue
  Inbound path policy configured
  Route map for incoming advertisements is testing
  Connections established 2; dropped 1
Connection state is ESTAB, I/O status: 1, unread input bytes: 0
Local host: 172.16.1.1, Local port: 12342
Foreign host: 172.16.1.2, Foreign port: 179

Enqueued packets for retransmit: 0, input: 0, saved: 0

Event Timers (current time is 0x530C294):
Timer          Starts    Wakeups            Next
Retrans            15          0             0x0
TimeWait            0          0             0x0
AckHold            15         13             0x0
SendWnd             0          0             0x0
KeepAlive           0          0             0x0
GiveUp              0          0             0x0
PmtuAger            0          0             0x0

<snip>
```

To view the status of a *specific* BGP neighbor:

RouterB# *show ip bgp neighbors 172.16.1.2*

* * *

BGP Synchronization



Consider the above example. AS 200 is serving as a **transit** between AS 100 and AS 300. BGP follows a synchronization rule that states that *all* routers in a transit AS, *including* non-BGP routers, must learn of a route before BGP can advertise it to an external peer.

Confused?

Consider the above example again. If RouterA advertises a BGP route to RouterB (an eBGP peer) for the 10.5.0.0/16 network, that same BGP route will eventually be forwarded to RouterD (an iBGP peer).

However, a **blackhole** would exist if RouterD then advertised that update to RouterE, as RouterC would not have the 10.5.0.0/16 network in its routing table. If RouterE attempts to reach the 10.5.0.0 network, RouterC will drop the packet.

BGP's synchronization rule will force RouterD to wait until RouterC learns the 10.5.0.0/16 route, before forwarding that route to RouterE. How will RouterD know when RouterC learns the route? Simple! When it receives an update from RouterC via an IGP (such as OSPF), containing that route.

BGP synchronization can be disabled under two circumstances:

- The local AS is not a transit between two other AS's
- All routers in the transit AS run iBGP, and are fully meshed.

To disable BGP synchronization:

```
RouterD(config)#  router bgp 200
RouterD(config-router)#  no synchronization
```

As of IOS 12.2(8)T, synchronization is **disabled** by default.

* * *

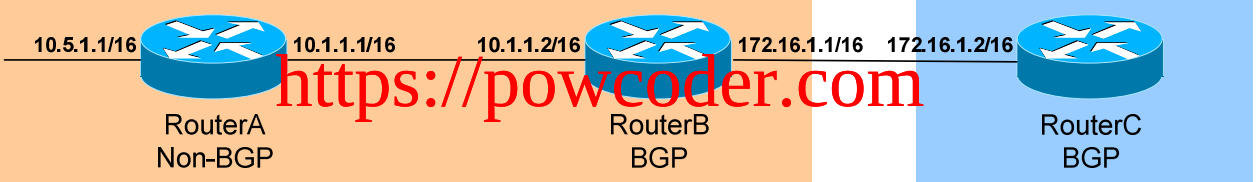
Originating Prefixes in BGP

There are three ways to originate a prefix (in other words, advertise a network) into BGP:

- By using *network* statements
- By using *aggregate-address* statements (explained later in this guide)
- By redistributing an IGP into BGP

Using the *network* statement informs BGP which networks to advertise to eBGP peers, *not* which interfaces to run BGP on. The *network* command can be used to inject *any* network from the local AS into BGP, include dynamic routes learned from an IGP, and not just the routes directly connected to the router.

However, the route ***must* be in the routing table** before BGP will advertise the network to an eBGP peer. **This is a fundamental BGP rule.**



Consider the above example. RouterB may inject the 10.5.0.0/16 network into BGP using the *network* command. However, unless that route is in the local routing table (in this case, via an IGP), RouterB will *not* advertise the route to RouterC.

Furthermore, the *network* statement **must match the route exactly** as it is in the routing table:

```
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 remote-as 900
RouterB(config-router)# network 10.5.0.0 mask 255.255.0.0
```

The above configuration would match the route perfectly, while the following configuration would not:

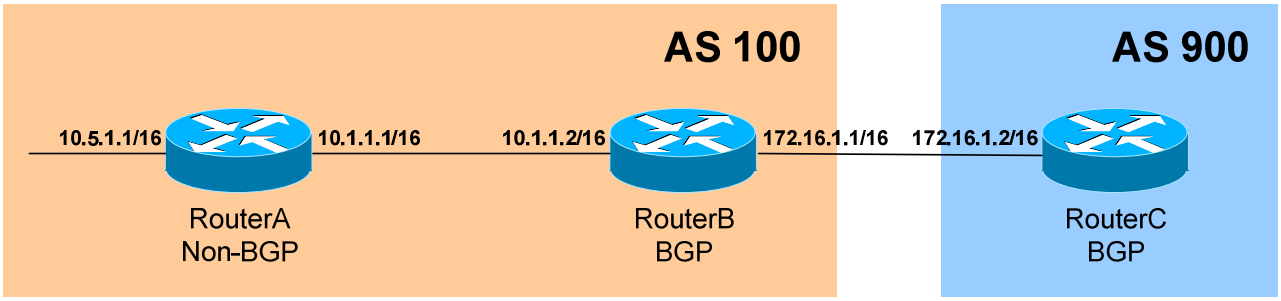
```
RouterB(config-router)# network 10.5.0.0 mask 255.255.255.0
```

If no mask is specified, a **classful** mask will be assumed.

* * *

The BGP Routing Table

Recall that BGP maintains its own **separate routing table**. This table contains a list of routes that can be advertised to BGP peers.



To view the BGP routing table on RouterB:

RouterB# *show ip bgp*

```
BGP table version is 426532, local router ID is 2.2.2.2
Status codes: s suppressed, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.5.0.0         0.0.0.0                  0     0      32768   i
```


The route has been injected into BGP using the *network* command. The *Next Hop* of *0.0.0.0* indicates that the route was locally originated into BGP. The *Path* is empty, as the route originated in the Autonomous Systems.

Notice the Status Codes of "*>". The * indicates that this route is *valid* (i.e. in the routing table). The > indicates that this is the *best* route to the destination.

BGP will **never** advertise a route to an eBGP peer unless it is both *valid* and the *best* route to that destination. BGP routes that are both *valid* and *best* will also added the **IP routing table** as well.

To view the BGP routing table on RouterC:

RouterC# *show ip bgp*

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 10.5.0.0         172.16.1.1               0     100      0    100 i
```

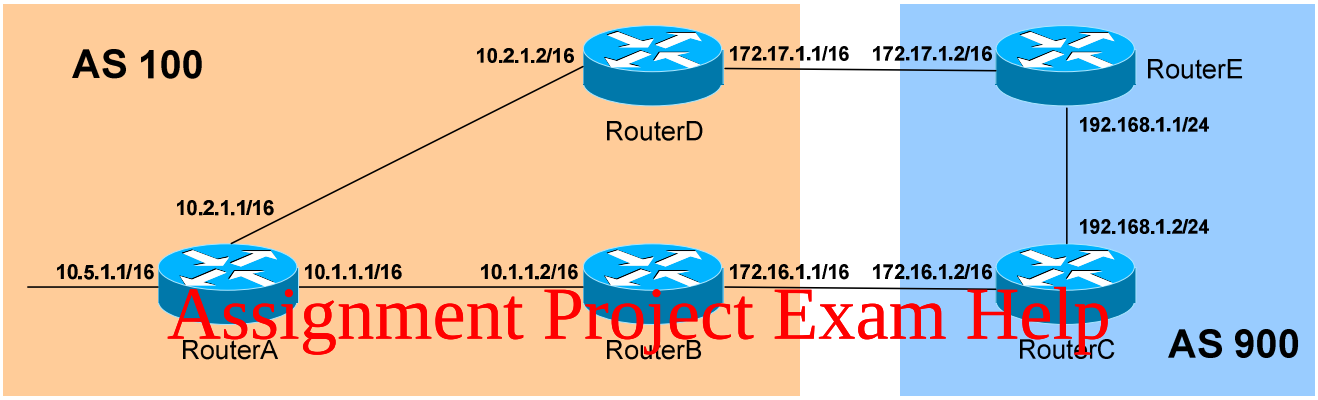
Notice that AS 100 has been added to the path, and that the Next Hop is now RouterB.

* * *

BGP Route-Reflectors

Recall that BGP requires all iBGP peers to be fully meshed. **Route-Reflectors** allow us to bypass this restriction. Fewer neighbor connections will result in less bandwidth and CPU usage.

Route-reflector **clients** form neighbor adjacencies with the route-reflector **server**. BGP updates will flow from the server to the clients, without the clients having to interact with each other.



Consider the above example. In AS 100, there are three BGP speakers. Normally, these iBGP peers *must* be fully-meshed. For example, RouterB would need a *neighbor* statement for both RouterA and RouterD.

As an alternative, RouterA can be configured as a route-reflector server. Both RouterB and RouterD would *only* need to peer with RouterA.

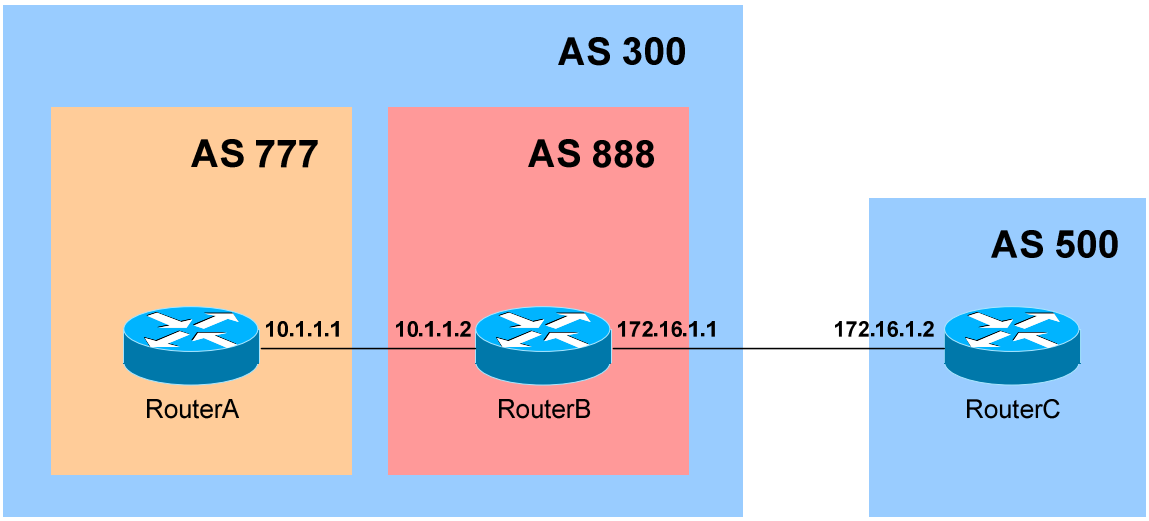
All route-reflector specific configuration takes place on the route reflector server:

```
RouterA(config)# router bgp 100
RouterA(config-router)# neighbor 10.2.1.2 remote-as 100
RouterA(config-router)# neighbor 10.2.1.2 route-reflector-client
RouterA(config-router)# neighbor 10.1.1.2 remote-as 100
RouterA(config-router)# neighbor 10.1.1.2 route-reflector-client
```

Route-reflectors are Cisco's recommended method of alleviating the iBGP full-mesh requirement.

* * *

BGP Confederations



Confederations are an alternative method to alleviate the requirement that all iBGP routers be fully meshed. Confederations are essentially AS's within an AS, and are sometimes referred to as **sub-AS's**.

In the above example, RouterA belongs to AS 777 and RouterB belongs to AS 888. Both of those AS's belong to a **parent** AS of 300. RouterA and RouterB will form an **eBGP** peer session.

Configuration is simple:


```
RouterB(config)# router bgp 888
RouterB(config-router)# bgp confederation identifier 300
RouterB(config-router)# bgp confederation peer 777
RouterB(config-router)# neighbor 10.1.1.1 remote-as 777
RouterB(config-router)# neighbor 172.16.1.2 remote-as 500
```

Notice that the sub-AS (*777*) is used in the *router bgp* statement. Additionally, the parent AS must be specified using a *bgp confederation identifier* statement. Finally, any *confederation peers* must be identified.

RouterC will be unaware of RouterB's confederation status. Thus, RouterC's neighbor statement will point to AS 300, and not AS 888:

```
RouterC(config)# router bgp 500
RouterC(config-router)# neighbor 172.16.1.1 remote-as 300
```

(Reference: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ics/icsbgp4.htm#wp6834)

* * *

BGP Peer-Groups

Peer-groups simplify configuration of groups of neighbors, assuming those neighbors share identical settings. Additionally, peer-groups conserve processor/memory resources by sending updates to all peer-group members *simultaneously*, as opposed to sending *individual* updates to each neighbor.

All neighbor parameters are applied to the peer-group itself. Configuration is simple:

```
Router(config)# router bgp 200
Router(config-router)# neighbor MYPEERGROUP peer-group
Router(config-router)# neighbor MYPEERGROUP remote-as 200
Router(config-router)# neighbor MYPEERGROUP update-source lo0
Router(config-router)# neighbor MYPEERGROUP route-reflector-client
```

The above configuration creates a peer-group named *MYPEERGROUP*, and applies the desired settings. Next, we must "assign" the appropriate neighbors to the peer-group:


```
Router(config-router)# neighbor 10.10.1.1 peer-group MYPEERGROUP
Router(config-router)# neighbor 10.10.2.2 peer-group MYPEERGROUP
Router(config-router)# neighbor 10.10.3.3 peer-group MYPEERGROUP
```


The above neighbors now inherit the settings of the peer-group named *MYPEERGROUP*.

All "members" of a peer-group must *exclusively* be internal (iBGP) peers *or* external (eBGP) peers. A mix of internal and external peers is not allowed in a peer-group.

Outbound route filtering (via a distribution-list, route-map, etc.) must be identical on all members of a peer-group. Inbound route filtering can still be applied on a per-neighbor basis.

(Reference: http://www.cisco.com/warp/public/459/29.html)

* * *

BGP Attributes

BGP utilizes several **attributes** to determine the best path to a destination.

Well-known attributes are supported by all implementations of BGP, while **optional** attributes may *not* be supported by all BGP-speaking routers.

Several **subcategories** of attributes exist:

- **Well-known Mandatory –** Standard attributes supported by all BGP implementations, and *always* included in every BGP update.
- **Well-known Discretionary –** Standard attributes supported by all BGP implementations, and are *optionally* included BGP updates.
- **Optional Transitive –** Optional attribute that may not be supported by all implementations of BGP. *Transitive* indicates that a non-compliant BGP router will forward the unsupported attribute unchanged, when sending updates to peers.
- **Optional Non-Transitive -** Optional attribute that may not be supported by all implementations of BGP. *Non-Transitive* indicates that a non-compliant BGP router will strip out the unsupported attribute, when sending updates to peers.

* * *

BGP Attributes (continued)

The following describes several **specific** BGP attributes:

- **AS-Path (well-known mandatory)** – Identifies the list (or path) of traversed AS's to reach a particular destination.
- **Next-Hop (well-known mandatory)** – Identifies the next hop IP address to reach a particular destination.
- **Origin (well-known mandatory) –** Identifies the originator of the route.
- **Local Preference (well-known, discretionary) –** Provides a preference to determine the best path for *outbound* traffic.
- **Atomic Aggregate (well-known discretionary) –** Identifies routes that have been summarized, or *aggregated*.
- **Aggregator (optional transitive) –** Identifies the BGP router that performed an address aggregation.
- **Community (optional transitive) –** Tags routes that share common characteristics into *communities*.
- **Multi-Exit-Discriminator (MED) (optional non-transitive) –** Provides a preference to eBGP peers to a specific *inbound* router.
- **Weight (Cisco Proprietary) –** Similar to Local Preference, provides a *local* weight to determine the best path for outbound traffic.

Each attribute is identified by a **code**:

Origin	Code 1
AS-Path	Code 2
Next Hop	Code 3
MED	Code 4
Local Preference	Code 5
Automatic Aggregate	Code 6
Aggregator	Code 7
Community	Code 8

* * *

BGP "Best Path" Determination

If BGP contains multiple routes to the same destination, it compares the routes in **pairs**, starting with the **newest** entries (listed higher in the routing table), and working towards the **oldest** entries (listed lower in the table).

BGP determines the best path by successively comparing the attributes of each "route pair." The attributes are compared in a specific order:

- **Weight –** Which route has the *highest* weight?
- **Local Preference –** Which route has the *highest* local preference?
- **Locally Originated –** Did the local router originate this route? In other words, is the next hop to the destination 0.0.0.0?
- **AS-Path –** Which route has the *shortest* AS-Path?
- **Origin Code –** Where did the route originate? The following origin codes are listed in order of preference:
  - IGP (originated from an interior gateway protocol)
  - EGP (originated from an exterior gateway protocol)
  - ? (Unknown origin)
- **MED –** Which path has the *lowest* MED?
- **BGP Route Type –** Is this an *eBGP* or *iBGP* route? (eBGP routes are preferred)
- **Age –** Which route is the oldest? (oldest is preferred)
- **Router ID –** Which route originated from the router with the lowest BGP router ID?
- **Peer IP Address –** Which route originated from the router with the lowest IP?

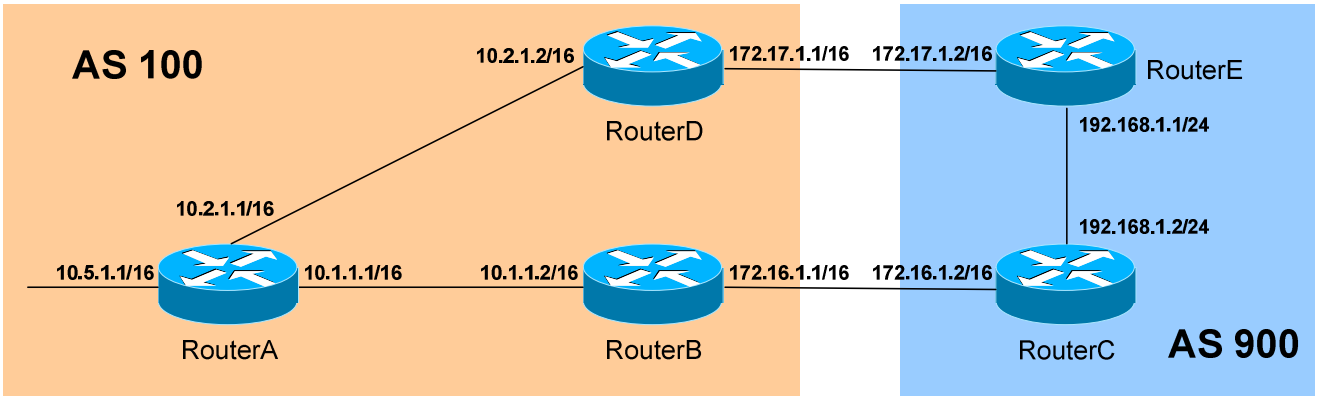
When applying attributes, Weight and Local Preference are applied to *inbound* routes, dictating the best *outbound* path.

AS-Path and MED are applied to *outbound* routes, dictating the best *inbound* path.

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml)

* * *

Weight



The **Weight** attribute is applied to *inbound* routes, dictating the best *outbound* path. It is a Cisco-proprietary attribute, and is only **locally significant** (and thus, is *never* passed on to BGP neighbors).

The weight value can range from 0 – 65535, and the **highest** weight is preferred. By default, a route originated on the local router will be assigned a weight of **32768**. All other routes will be assigned a weight of **0**, by default.

A weight value can be specified for *all routes* advertised from a specific neighbor:


```
RouterA(config)# router bgp 100
RouterA(config)# neighbor 10.1.1.2 weight 200
```


Otherwise, a weight value can be specified for *specific routes* from a particular neighbor. First, the prefixes in question must be identified:

```
RouterA(config)# ip prefix-list MYLIST 192.168.1.0/24
```

Then, a route-map is used to apply the appropriate weight:

```
RouterA(config)# route-map WEIGHT permit 10
RouterA(config-route-map)# match ip address prefix-list MYLIST
RouterA(config-route-map)# set weight 200
RouterA(config-route-map)# route-map WEIGHT permit 20
```

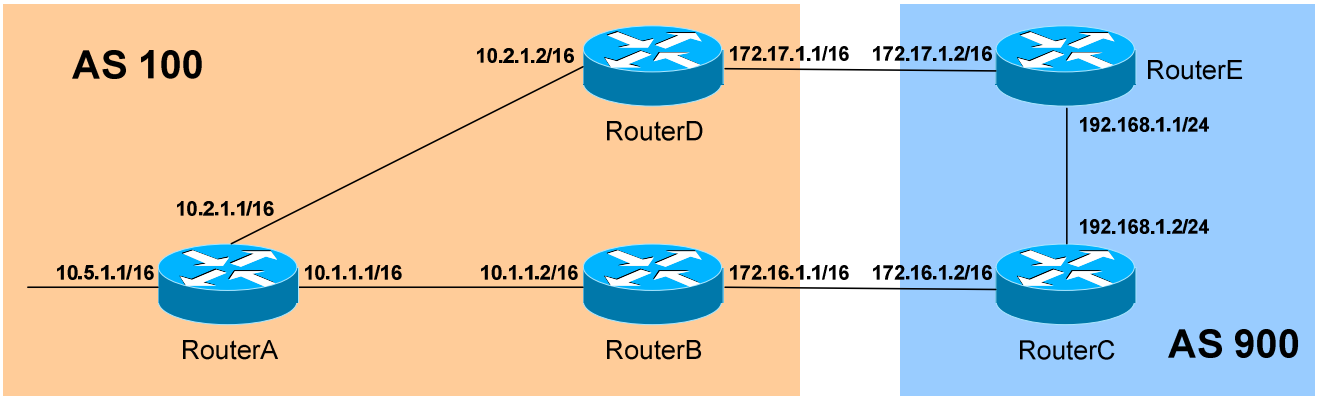
Finally, the route-map is applied to the preferred neighbor:

```
RouterA(config)# router bgp 100
RouterA(config)# neighbor 10.1.1.2 route-map WEIGHT in
```

(Reference: http://www.cisco.com/warp/public/459/bgp-toc.html#weight)

* * *

Local Preference



The **Local Preference** attribute is applied to *inbound* external routes, dictating the best *outbound* path. Unlike the Weight attribute, Local Preference is passed on to **iBGP** peers when sending updates. Local Preference informs iBGP routers how to *exit* the AS, if multiple paths exist.

Local Preference is a 32-bit number, and can range from 0 to 4294967295. The **highest** Local Preference is preferred, and the default preference is **100.**

The Local Preference value can be specified for *all inbound external routes*, on a global basis for BGP:


```
RouterB(config)# router bgp 100
RouterB(config-router)# bgp default local-preference 200
```

```
RouterD(config)# router bgp 100
RouterD(config-router)# bgp default local-preference 300
```

Both RouterB and RouterD will include the Local Preference attribute in updates to iBGP neighbors. Thus, RouterA (*and* RouterB) will now prefer the route through RouterD to reach any destination outside the local AS.

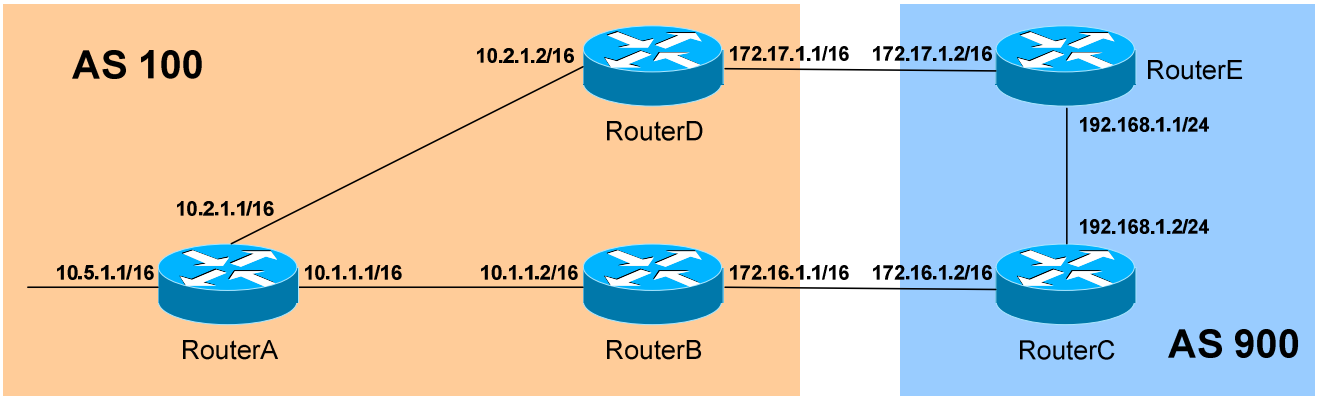
Local Preference can be applied on a per-route basis:

```
RouterD(config)# ip prefix-list MYLIST 192.168.1.0/24

RouterD(config)# route-map PREFERENCE permit 10
RouterD(config-route-map)# match ip address prefix-list MYLIST
RouterD(config-route-map)# set local-preference 300

RouterD(config)# router bgp 10
RouterD(config)# neighbor 172.17.1.2 route-map PREFERENCE in
```

(Reference: http://www.cisco.com/warp/public/459/bgp-toc.html#localpref)

* * *

AS-Path Prepend



The **AS-Path attribute** is applied to *outbound* routes, dictating the best *inbound* path. Two things can be accomplished with the AS-Path attribute, *prepend* or *filter*.

To *prepend* to (or add to) the existing AS-Path results in a *longer* AS-Path, which makes the route less desirable for inbound traffic:


```
RouterB(config)# access-list 5 permit 10.5.0.0 0.0.255.255
RouterB(config)# route-map ASPREPEND permit 10
RouterB(config-route-map)# match ip address 5
RouterB(config-route-map)# set as-path prepend 200 200
RouterB(config-route-map)# route-map ASPREPEND permit 20
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 route-map ASPREPEND out
```

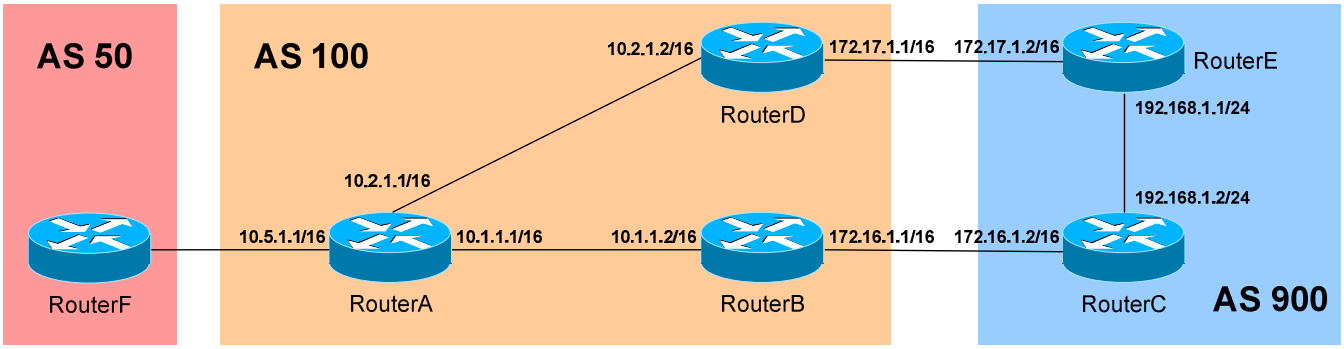
The artificial AS-Path information is not added to a route until it is advertised to an eBGP peer. RouterC's BGP routing table will now look as follows:

```
RouterC# show ip bgp

   Network          Next Hop        Metric LocPrf Weight Path
*  10.5.0.0         172.16.1.1         0     100     0     100 200 200 i
*> 10.5.0.0         172.17.1.1         0     100     0     100 i
```

Notice the inflated AS-Path through RouterB. RouterC will prefer the path through RouterD to reach the 10.5.0.0/16 network.

* * *

AS-Path Filtering



Additionally, routes can be *filtered* based on AS-Path values, using an *as-path access-list*. This requires the use of **regular expressions:**

- ^ = Start of a string
- **$** = End of a string
- **.** = Any one character
- * = Any one or more characters, including none
- + = Any one or more characters
- ? = Any one character, including none
- _ = Serves the function of virtually all of the above

The following examples illustrate the use of regular expressions:

- ^100_ = learned from AS 100
- _100$ = originated from AS 100
- ^$ = originated locally
- .* = matches everything
- _100_ = any instance of AS 100

To configure RouterF to only accept routes that *originated* from AS100:

RouterF(config)# *ip as-path access-list 15 permit _100$*

RouterF(config)# *route-map ASFILTER permit 10*
RouterF(config-route-map)# *match as-path 15*

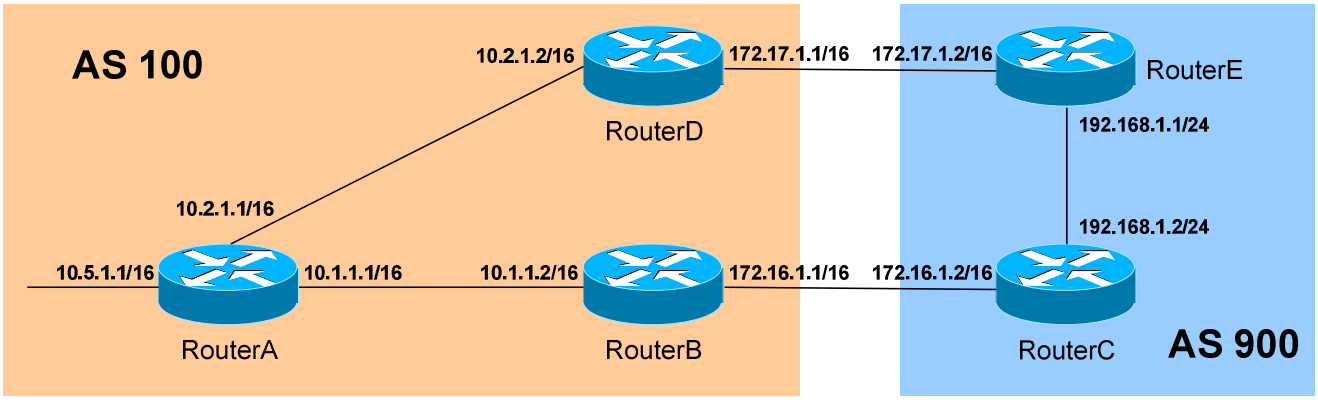
RouterF(config)# *router bgp 50*
RouterF(config-router)# *neighbor 10.5.1.1 route-map ASFILTER in*

To view what BGP routing entries the AS-Path access-list will match:

RouterF# *show ip bgp regexp _100$*

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094a92.shtml)

Origin



The **Origin attribute** identifies the originating *source* of the route. The origin codes are as follows (listed in order of preference for route selection):

- **i (IGP) –** Originated from an interior gateway protocol, such as OSPF. This usually indicates the route was injected into BGP via the *network* command under the BGP process. An origin code of **"i"** is most preferred.
- **e (EGP)** - Originated from an external gateway protocol.
- **? (incomplete)** - Unknown origin. This usually indicates the route was redistributed into BGP (from either connected, static, or IGP routes). An origin code of **"?"** is the least preferred.

When viewing the BGP routing table, the origin code is listed at the *end* of each line in the table:

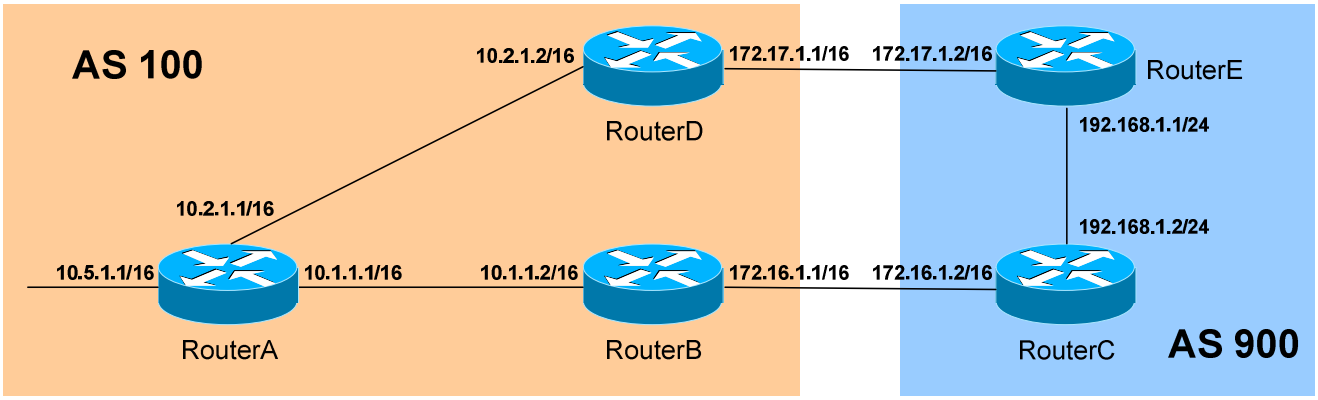
RouterB# *show ip bgp*

```
   Network          Next Hop        Metric LocPrf Weight Path
*> 10.5.0.0         10.1.1.1             0     0      0     i
*> 192.168.1.0      172.16.1.2           0     100    0     900 ?
```

The *i* at the end of the first routing entry indicates the *10.5.0.0* network was originated via an IGP, probably with the BGP *network* command. The *192.168.1.0* network was most likely redistributed into BGP in AS 900, as evidenced by the *?* at the end of that routing entry.

* * *

MED



The **MED (MultiExit Discriminator) attribute** is applied to *outbound* routes, dictating the best *inbound* path into the AS (assuming multiple paths exist). The MED is identified as the BGP **metric** when viewing the BGP routing table. A **lower** metric is preferred, and the default MED value is **0**.

In the above example, there are two entry points into AS 100. To force AS 900 to prefer that path through RouterD to reach the 10.5.0.0/16 network, the *set metric* command can be used with a route-map:

RouterB(config)# *access-list 5 permit 10.5.0.0 0.0.255.255*

RouterB(config)# *route-map SETMED permit 10*
RouterB(config-route-map)# *match ip address 5*
RouterB(config-route-map)# *set metric 100*

RouterB(config)# *router bgp 100*
RouterB(config-router)# *neighbor 172.16.1.2 route-map SETMED out*

RouterC will now have two entries for the 10.5.0.0/16 route:

RouterC# *show ip bgp*

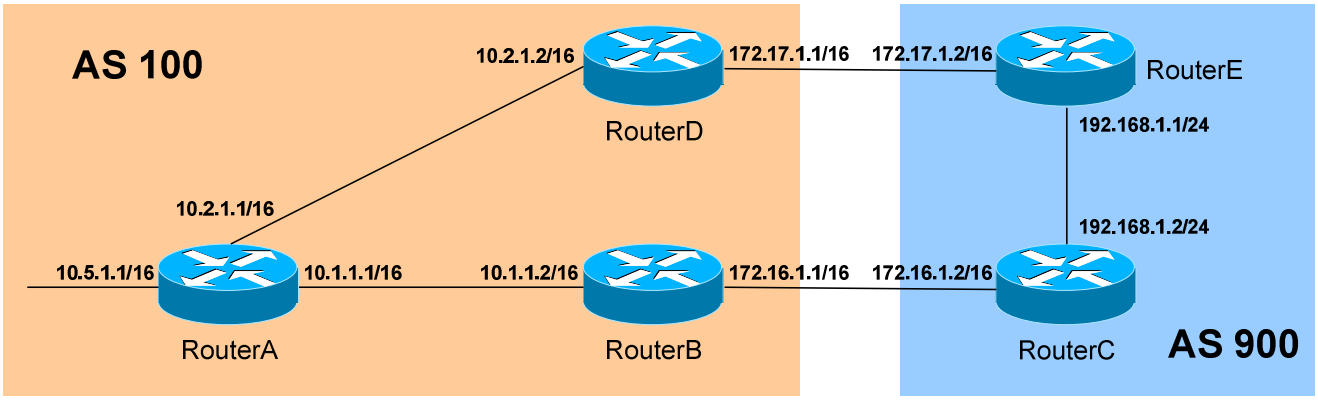
```
   Network          Next Hop        Metric LocPrf Weight Path
*  10.5.0.0         172.16.1.1         200    100      0   100 i
*> 10.5.0.0         172.17.1.1           0    100      0   100 i
```

Notice that the route from RouterB has a higher metric, and thus is less preferred. Note specifically the lack of a > on the route with a higher metric.

The MED value is exchanged from one AS to another, but will *never* be advertised further than that. Thus, the MED value is passed from AS 100 to all BGP routers in AS 900, but the metric will be reset to **0** if the route is advertised beyond AS 900.

* * *

MED (continued)



A key aspect to consider when using the MED attribute is BGP's method of route selection. Recall that if BGP contains multiple routes to the same destination, it compares the routes in **pairs**, starting with the **newest** entries and working towards the **oldest** entries.

This can lead to sub-optimal routing, depending on the order of routes in the BGP routing table. BGP employs two MED-related commands to alleviate potential sub-optimal routing selections.

The ***bgp deterministic-med*** command forces the MED value to be compared, when multiple routes to the same network are received via *multiple routers from the **same** AS*, regardless of the order of routes in the BGP routing table.


```
RouterE(config)#  router bgp 100
RouterE(config-router)#  bgp deterministic-med
```

The *bgp deterministic-med* command is disabled by default. If used, the command should be enabled on *all* routers within the AS.

The ***bgp always-compare-med*** command forces the MED value to be compared, when multiple routes to the same network are received via *multiple routers from **different** AS's*, regardless of the order of routes in the BGP routing table.

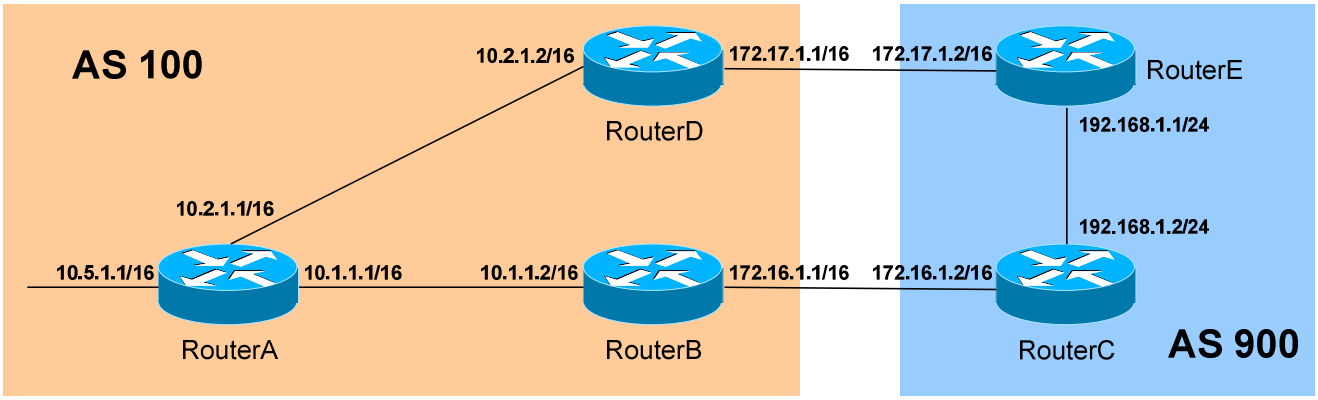
```
RouterE(config)#  router bgp 100
RouterE(config-router)#  bgp always-compare-med
```

The *bgp always-compare-med* command is disabled by default. Thus, by default, the MED value is *not* compared between paths from different AS's.

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094925.shtml; http://www.cisco.com/warp/public/459/bgp-toc.html#metricattribute)

* * *

MED (continued)



The MED metric on routes sent to eBGP neighbors can be dynamically set to the actual metric of an IGP (such as OSPF). This is accomplished using the *set metric-type internal* command with a route-map:

```
RouterB(config)# access-list 5 permit 10.5.0.0 0.0.255.255
RouterB(config)# route-map MED_INTERNAL permit 10
RouterB(config-route-map)# match ip address 5
RouterB(config-route-map)# set metric-type internal
RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.17.1.2 route-map MED_INTERNAL out
```


If the *10.5.0.0/16* network originated in OSPF, the link-state **cost** metric for that route will be applied as the MED metric.

* * *

Communities

BGP allows routes to be placed (or *tagged*) into certain **Communities.** BGP routers can make route policy decisions based on a route's community membership.

BGP communities can be assigned using one of three **32-bit** formats:

- **Decimal** (1000000)
- **Hexadecimal** (0x1A2B3C)
- **AA:NN** (100:20)

The AA:NN format specifies a 16-bit AS number (the AA), and a 16-bit generic community identifier (NN).

By default, the *decimal* format for communities will be displayed when viewing a route. To force the router to display the AA:NN format:

RouterA(config)# *ip bgp-community new-format*

Additionally, there are four **well-known** communities that can be referenced by name:

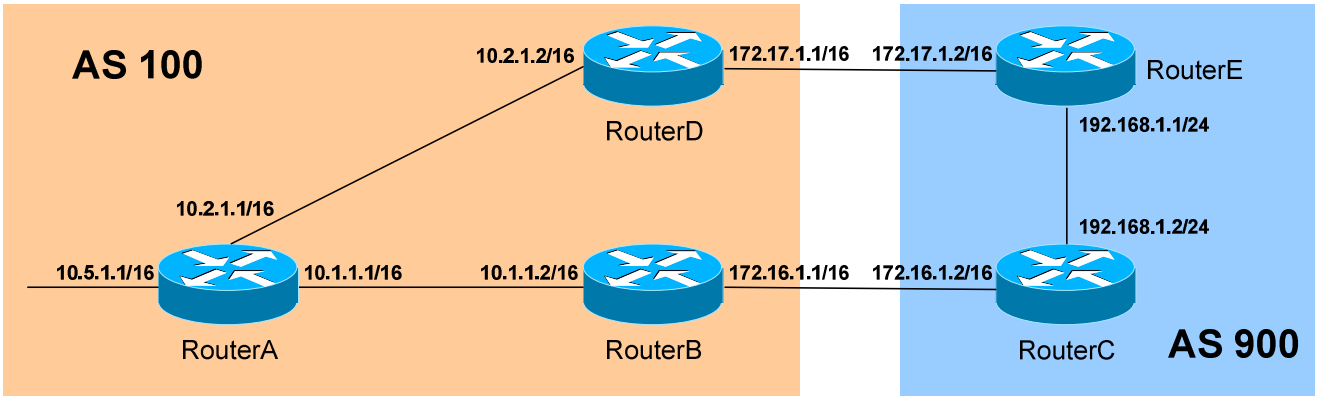
- **No-export** – prevents the route from being advertised outside the local AS to eBGP peers.
- **No-advertise** – prevents the route from being advertised to either internal or external peers.

- **Internet** – allows the route to be advertised outside the local AS.
- **Local-AS –** prevents the route from being advertised outside the local AS to either eBGP *or* confederate peers.

(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_q_and_a_item09186a00800949e8.shtml#four; http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800c95bb.shtml#communityattribute)

* * *

Communities (continued)



To set the community for a specific route, using a route-map:

```
RouterB(config)# access-list 5 permit 10.5.0.0 0.0.255.255

RouterB(config)# route-map COMMUNITY permit 10
RouterB(config-route-map)# match ip address 5
RouterB(config-route-map)# set community no-export
RouterB(config)# route-map COMMUNITY permit 20

RouterB(config)# router bgp 100
RouterB(config-router)# neighbor 172.16.1.2 send-community
RouterB(config-router)# neighbor 172.16.1.2 route-map COMMUNITY out
```


The community attribute will not be advertised to a neighbor unless the *send-community* parameter is applied to the *neighbor* command, regardless if a community value is applied using a route-map.

The above configuration will place the *10.5.0.0/16* route into the *no-export* community once it is advertised into AS 900. RouterC will advertise this network to all iBGP peers, but the community attribute will prevent RouterC (and all iBGP peers) from advertising the route outside of AS 900.

By default, the *set community* route-map command will **overwrite** any existing community parameters for a route. To instead **append** additional community values, the *additive* parameter must be specified:

```
RouterB(config)# route-map COMMUNITY permit 10
RouterB(config-route-map)# match ip address 5
RouterB(config-route-map)# set community no-export additive
RouterB(config)# route-map COMMUNITY permit 20
```

* * *

BGP Summarization

Routes that are **redistributed** into BGP are **automatically summarized**. To disable auto-summary:

Router(config)# *router bgp 100*
Router(config-router)# *no auto-summary*

To manually create a summary address for the following group of networks:

- 172.16.0.0/24
- 172.16.1.0/24
- 172.16.2.0/24
- 172.16.3.0/24

The *aggregate-address* command must be used:

Router(config)# *router bgp 100*
Router(config-router)# *aggregate-address 172.16.0.0 255.255.252.0*

BGP's default configuration is to send ***both*** the summary (or aggregated) address ***and*** the more specific individual routes. To *only* send the summary route:

Router(config)# *router bgp 100*
Router(config-router)# *aggregate-address 172.16.0.0 255.255.252.0 summary-only*

To **suppress** (or *summarize*) only specific routes, instead of all routes, a route-map must be used:

Router(config)# *access-list 5 permit 172.16.0.0 0.0.0.255*
Router(config)# *access-list 5 permit 172.16.1.0 0.0.0.255*

Router(config)# *route-map SUPPRESS permit 10*
Router(config-route-map)# *match ip address 5*

Router(config)# *router bgp 100*
Router(config-router)# *aggregate-address 172.16.0.0 255.255.252.0 summary-only suppress-map SUPPRESS*

The access-list details the routes that *should* be suppressed. To allow the summarized routes to retain their AS-Path information:

Router(config)# *router bgp 100*
Router(config-router)# *aggregate-address 172.16.0.0 255.255.252.0 summary-only suppress-map SUPPRESS as-set*

* * *

BGP Route Dampening

Route dampening "suppresses" routes that are flapping, minimizing unnecessary convergence and updates. If a route **flaps** (goes up and down), it is assigned a penalty (default is **1000**). All routes start with a penalty of 0, and the local router maintains a history of routes that have flapped.

Once the penalty reaches a specific threshold, the route is suppressed. When a route is suppressed, it is neither advertised nor used locally on the router.

First, the routes to be "observed" must be identified using an access-list or prefix-list:

Router(config)# *ip prefix-list MYLIST seq 10 permit 10.1.0.0/16*
Router(config)# *ip prefix-list MYLIST seq 20 permit 10.2.0.0/16*

Next, dampening values must be configured using a route-map:

Router(config)# *route-map MYMAP permit 10*
Router(config-route-map)# *match ip address prefix-list MYLIST*
Router(config-route-map)# *set dampening 15 750 2000 60*

The above values for the *set dampening* command represent the defaults.

The *15* (measured in minutes) indicates the half-life timer. If a route is assigned a penalty, half of the penalty will decay after this timer expires.

The *750* (arbitrary penalty measurement) indicates the bottom threshold. Once a penalized route falls below this threshold, it will no longer be suppressed.

The *2000* (arbitrary penalty measurement) indicates the top threshold. If a route flaps to the point that its penalty exceeds this threshold, it is suppressed.

The *60* (measured in minutes) indicates the maximum amount of time a route can be suppressed.

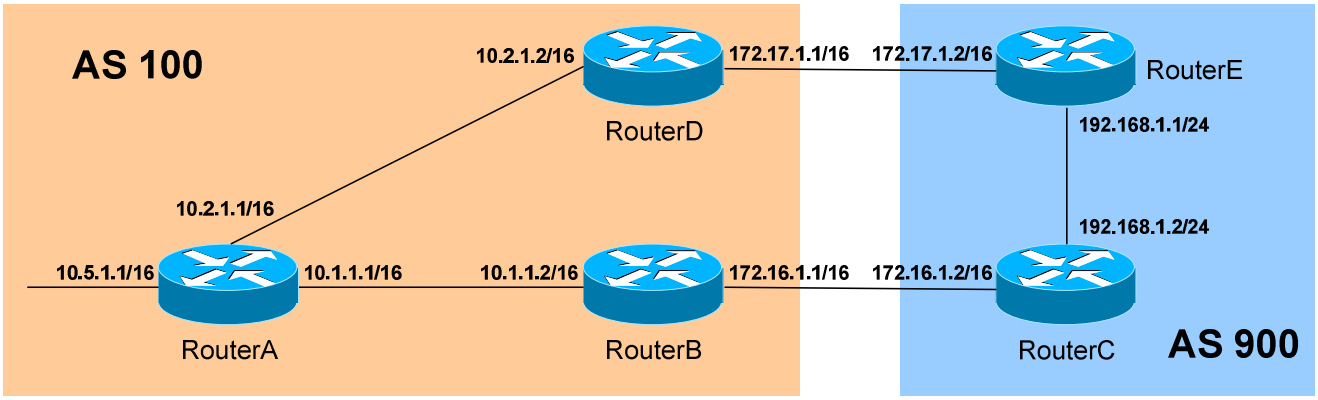
Finally, route-dampening must be enabled under the BGP process:

Router(config)# *router bgp 100*
Router(config-router)# *bgp dampening route-map MYMAP*

(Reference: http://www.cisco.com/warp/public/459/bgp-rec-routing.html)

* * *

BGP Next-Hop-Self



Consider the above diagram. If RouterC sends the 192.168.1.0/24 route to its eBGP peer RouterB, the Next Hop for that route will be through RouterC:

RouterB# *show ip bgp*

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      172.16.1.2               0    100      0 900 i
```

A serious problem arises when RouterB sends this route to its iBGP peers (RouterA and RouterD). The Next Hop value is *not* changed:

RouterA# *show ip bgp*

```
   Network          Next Hop            Metric LocPrf Weight Path
*  192.168.1.0      172.16.1.2               0    100      0 900 i
```

Notice the lack of >, indicating this is no longer the *best* route to the destination. This is because RouterA has no route to the next hop address.

There are two workarounds. Either the 172.16.0.0/16 network must be added to RouterA's and RouterD's routing tables, or the Next-Hop field must be adjusted to identify RouterB as the next hop.

The configuration is simple, and is completed on RouterB:

RouterB(config)# *router bgp 200*
RouterB(config-router)# *neighbor 10.1.1.1 next-hop-self*
RouterB(config-router)# *neighbor 10.2.1.2 next-hop-self*

RouterB now advertises itself as the next hop for all eBGP routes it learns:

RouterA# *show ip bgp*

```
   Network          Next Hop            Metric LocPrf Weight Path
*> 192.168.1.0      10.1.1.2                 0    100      0 900 i
```

* * *

BGP Backdoor

Recall that an external BGP route has an Administrative Distance (AD) of **20**, which is less than the default AD of IGP's, such as OSPF or EIGRP.

Under certain circumstances, this may result in sub-optimal routing. If both an IGP route and eBGP route exist to the same network, and the IGP route *should* be preferred, there are two workarounds:

- Globally change BGP's default Administrative Distance values.
- Use the BGP *network backdoor* command.

Cisco does not recommend changing BGP's default AD values. If necessary, however, the *distance bgp* will adjust the AD for **external, internal**, and **locally-originated** BGP routes, respectively:

```
Router(config)# router bgp 100
Router(config-router)# distance bgp 150 210 210
```


The preferred workaround is to use the BGP *network backdoor* command, which adjusts the AD for a specific eBGP route (by default, from **20** to **200)**, resulting in the IGP route being preferred:


```
Router(config)# router bgp 100
Router(config-router)# network 10.5.0.0 mask 255.255.0.0 backdoor
```


(Reference: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a00800c95bb.shtml#bgpbackdoor)

* * *

Misc. BGP Commands

To restrict the number of routes a BGP router can receive from its neighbor:

Router(config)# *router bgp 200*
Router(config-router)# *neighbor 10.1.1.1 maximum-prefix 10000*

To immediately reset an eBGP session if a link connecting two peers goes down, the *bgp fast-external-fallover* feature must be enabled. To enable this feature globally:

Router(config)# *router bgp 200*
Router(config-router)# *bgp fast-external-fallover*

To enable this feature on a per-interface basis:

Router(config)# *int serial0/0*
Router(config-if)# *ip bgp fast-external-fallover permit*

To reset the BGP session between all neighbors:

Router# *clear ip bgp **

To force a resend of routing updates, *without* resetting any BGP sessions between neighbors:

Router# *clear ip bgp * soft*

To view a summary of all BGP connections, including the total number of BGP routes and a concise list of neighbors:

Router# *show ip bgp summary*

* * *