Individual routing algorithm components interact in the **control plane.**

Static routing algorithms, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a link costs).

Dynamic routing algorithms change the routing paths as the network traffic loads or topology change.

In a **load-sensitive algorithm**, link costs vary dynamically to reflect the current level of congestion in the underlying link.

If a high cost is associated with a link that is currently congested, a routing algorithm will tend to choose routes around such a congested link. **(T/F)**

In a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice this is accomplished by having each node broadcast link-state packets to *all* other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice (for example, with the Internet's OSPF routing protocol, discussed in **Section 5.3**) this is often accomplished by a **link-state broadcast** algorithm **[Perlman 1999]**. The result of the nodes' broadcast is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

Dijkstra's algorithm is iterative and has the property that after the kth iteration of the algorithm, the least-cost paths are known to k destination nodes, and among the least-cost paths to all destination nodes, these k paths will have the k smallest costs.

Autonomous systems (ASs), with each AS consisting of a group of routers that are under the same administrative control.

The routing algorithm running within an autonomous system is called an **intra-autonomous system routing protocol**.

OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra's least-cost path algorithm. With OSPF, each router constructs a complete topological map (that is, a graph) of the entire autonomous system. Each router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all *subnets*, with itself as the root node.

When routing a packet between a source and destination within the same AS, the route the packet follows is entirely determined by the intra-AS routing protocol. However, to route a packet across multiple ASs, say from a smartphone in Timbuktu to a server in a datacenter in Silicon Valley, we need an **inter-autonomous system routing protocol**.

Since an inter-AS routing protocol involves coordination among multiple ASs, communicating ASs must run the same inter-AS routing protocol. In fact, in the Internet, all ASs run the same inter-AS routing protocol, called the Border Gateway Protocol, more commonly known as **BGP [RFC 4271**; **Stewart 1999]**.

An IPv4 address is 32 bits long.
The IPv4 addresses are unique and universal.
The address space of IPv4 is 232 or 4,294,967,296.

Change the following IPv4 addresses from binary notation to dotted-decimal notation.
a. 10000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111
c. 11100111 11011011 10001011 01101111
d. 11111001 10011011 11111011 00001111

Find the error, if any, in the following IPv4 addresses:
a. 111.56.045.78
b. 221.34.7.8.20
c. 75.45.301.14
d. 11100010.23.14.67

Change the following IPv4 addresses from binary notation to hexadecimal notation.
a. 10000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111

Find the number of addresses in a range if the first address is 146.102.29.0 and the last address is 146.102.32.255.

The first address in a range of addresses is 14.11.45.96. If the number of addresses in the range is 32, what is the last address?

Find the class of each address:
a. 00000001 00001011 00001011 11101111
b. 11000001 10000011 00011011 11111111
c. 10100111 11011011 10001011 01101111
d. 11110011 10011011 11111011 00001111

An address in a block is given as 200.11.8.45. Find the number of addresses in the block, the first address, and the last address.

A router receives a packet with the destination address 201.24.67.32. Show how the router finds the network address of the packet.

What is the prefix length and suffix length if the whole Internet is considered as one single block with 4,294,967,296 addresses?

What is the prefix length and suffix length if the Internet is divided into 4,294,967,296 blocks and each block has one single address?

One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

First address = (any address) AND (network mask)
Last address = (any address) OR [NOT (network mask)]

One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

One of the addresses in a block is 17.63.110.114/24. Find the number of addresses, the first address, and the last address in the block.

One of the addresses in a block is 110.23.120.14/20. Find the number of addresses, the first address, and the last address in the block.

In classful addressing, the IPv4 address space is divided into five classes: A, B, C, D, and E. An organization is granted a block in one of the three classes, A, B, or C. Classes D and E is reserved for special purposes. An IP address in classes A, B, and C is divided into netid and hostid.

❑ In classful addressing, the first address in the block is called the network address. It defines the network to which an address belongs. The network address is used in routing a packet to its destination network.

❑ A network mask or a default mask in classful addressing is a 32-bit number with n leftmost bits all set to 1s and (32 − n) rightmost bits all set to 0s. It is used by a router to find the network address from the destination address of a packet.

❑ The idea of splitting a network into smaller subnetworks is called subnetting. A subnetwork mask, like a network mask, is used to find the subnetwork address when a destination IP address is given. In supernetting, an organization can combine several class C blocks to create a larger range of addresses.

❑ In 1996, the Internet authorities announced a new architecture called classless addressing or CIDR that allows an organization to have a block of addresses of any size as long as the size of the block is a power of two.

Only data in a datagram is fragmented. (T/F)

A packet has arrived with an M bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

If the M bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.

If the M bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don't know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.

A packet has arrived with an M bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

A packet has arrived in which the offset value is 100, the value of HLEN is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

Table 10-2 RFC 1918 Private Address Space

Range of IP Addresses	Network(s)	Class of Networks	Number of Networks
10.0.0.0 to 10.255.255.255	10.0.0.0	A	1
172.16.0.0 to 172.31.255.255	172.16.0.0 – 172.31.0.0	B	16
192.168.0.0 to 192.168.255.255	192.168.0.0 – 192.168.255.0	C	256

1. Which of the following routing protocols is considered to use link-state logic?
   a. RIPv1
   b. RIPv2
   c. EIGRP
   d. OSPF
2. Which of the following routing protocols use a metric that is, by default, at least partially affected by link bandwidth? (Choose two answers.)
   a. RIPv1
   b. RIPv2
   c. EIGRP
   d. OSPF
3. Which of the following interior routing protocols support VLSM? (Choose three answers.)
   a. RIPv1
   b. RIPv2
   c. EIGRP
   d. OSPF

Routing protocol: A set of messages, rules, and algorithms used by routers for the overall purpose of learning routes. This process includes the exchange and analysis of routing information. Each router chooses the best route to each subnet (path selection) and finally places those best routes in its IP routing table. Examples include RIP, EIGRP, OSPF, and BGP.

■ **Routed protocol and routable protocol:** Both terms refer to a protocol that defines a packet structure and logical addressing, allowing routers to forward or route the packets. Routers forward packets defined by routed

Interior and Exterior Routing Protocols IP routing protocols fall into one of two major categories:

Interior gateway protocols (IGP) or exterior gateway protocols (EGP).

The definitions of each are as follows:

■ IGP: A routing protocol that was designed and intended for use inside a single autonomous system (AS)

■ EGP: A routing protocol that was designed and intended for use between different autonomous systems

Table 19-2 IP IGP Metrics

IGP	Metric	Description
RIPv2	Hop count	The number of routers (hops) between a router and the destination subnet
OSPF	Cost	The sum of all interface cost settings for all links in a route, with the cost defaulting to be based on interface bandwidth
EIGRP	Calculation based on bandwidth and delay	Calculated based on the route's slowest link and the cumulative delay associated with each interface in the route

Table 19-3 Interior IP Routing Protocols Compared

Feature	RIPv2	EIGRP	OSPF
Classless/sends mask in updates/supports VLSM	Yes	Yes	Yes
Algorithm (DV, advanced DV, LS)	DV	Advanced DV	LS
Supports manual summarization	Yes	Yes	Yes
Cisco-proprietary	No	Yes[1]	No
Routing updates are sent to a multicast IP address	Yes	Yes	Yes
Convergence	Slow	Fast	Fast

Table 19-4 Default Administrative Distances

Route Type	Administrative Distance
Connected	0
Static	1
BGP (external routes [eBGP])	20
EIGRP (internal routes)	90
IGRP	100
OSPF	110
IS-IS	115
RIP	120
EIGRP (external routes)	170
BGP (internal routes [iBGP])	200
DHCP default route	254
Unusable	255

Table 19-7 OSPF Design Terminology

Term	Description
Area Border Router (ABR)	An OSPF router with interfaces connected to the backbone area and to at least one other area
Backbone router	A router connected to the backbone area (includes ABRs)
Internal router	A router in one area (not the backbone area)
Area	A set of routers and links that shares the same detailed LSDB information, but not with routers in other areas, for better efficiency
Backbone area	A special OSPF area to which all other areas must connect—area 0
Intra-area route	A route to a subnet inside the same area as the router
Interarea route	A route to a subnet in an area of which the router is not a part

Table 19-8 The Three OSPFv2 LSA Types Seen with a Multiarea OSPF Design

LSA Name	LSA Type	Primary Purpose	Contents of LSA
Router	1	Describe a router	RID, interfaces, IP address/mask, current interface state (status)
Network	2	Describe a network that has a DR	DR and BDR IP addresses, subnet ID, mask
Summary	3	Describe a subnet in another area	Subnet ID, mask, RID of ABR that advertises the LSA

- Routing Information Protocol -

RIP (Routing Information Protocol)

RIP is a standardized Distance Vector protocol, designed for use on smaller networks. RIP was one of the first true Distance Vector routing protocols, and is supported on a wide variety of systems.

RIP adheres to the following Distance Vector characteristics:

- RIP sends out periodic routing updates (every **30 seconds**)
- RIP sends out the full routing table every periodic update
- RIP uses a form of distance as its metric (in this case, **hopcount**)
- RIP uses the Bellman-Ford Distance Vector algorithm to determine the best "path" to a particular destination

Other characteristics of RIP include:

- RIP supports IP and IPX routing.
- RIP utilizes UDP port 520
- RIP routes have an administrative distance of **120**.
- RIP has a maximum hopcount of **15 hops.**

Any network that is 16 hops away or more is considered unreachable to RIP, thus the maximum diameter of the network is 15 hops. A metric of 16 hops in RIP is considered a **poison route** or **infinity metric.**

If multiple paths exist to a particular destination, RIP will load balance between those paths (by default, up to **4**) only if the metric (hopcount) is **equal**. RIP uses a round-robin system of load-balancing between equal metric routes, which can lead to **pinhole congestion**.

For example, two paths might exist to a particular destination, one going through a 9600 baud link, the other via a T1. If the metric (hopcount) is equal, RIP will load-balance, sending an equal amount of traffic down the 9600 baud link and the T1. This will (obviously) cause the slower link to become congested.

* * *

RIP Versions

RIP has two versions, **Version 1** (**RIPv1**) and **Version 2** (**RIPv2**).

RIPv1 (RFC 1058) is **classful**, and thus does not include the subnet mask with its routing table updates. Because of this, RIPv1 does not support **Variable Length Subnet Masks (VLSMs**). When using RIPv1, networks must be contiguous, and subnets of a major network must be configured with identical subnet masks. Otherwise, route table inconsistencies (or worse) will occur.

RIPv1 sends updates as **broadcasts** to address 255.255.255.255.

RIPv2 (RFC 2543) is **classless**, and thus *does* include the subnet mask with its routing table updates. RIPv2 fully supports VLSMs, allowing discontiguous networks and varying subnet masks to exist.

Other enhancements offered by RIPv2 include:

- Routing updates are sent via **multicast**, using address **224.0.0.9**
- Encrypted authentication can be configured between RIPv2 routers
- Route tagging is supported (explained in a later section)

RIPv2 can interoperate with RIPv1. By default:

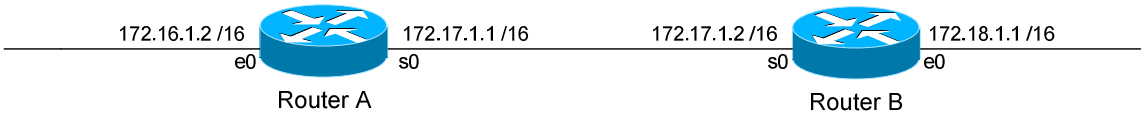
- RIPv1 routers will sent only Version 1 packets
- RIPv1 routers will receive both Version 1 and 2 updates
- RIPv2 routers will both send and receive only Version 2 updates

We can control the version of RIP a particular interface will "send" or "receive."

Unless RIPv2 is manually specified, a Cisco will default to RIPv1 when configuring RIP.

* * *

RIPv1 Basic Configuration



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure RIP, we would type:

Router(config)# *router rip*
Router(config-router)# *network 172.16.0.0*
Router(config-router)# *network 172.17.0.0*

The first command, *router rip*, enables the RIP process.

The *network* statements tell RIP which networks you wish to advertise to other RIP routers. We simply list the networks that are directly connected to our router. Notice that we specify the networks at their classful boundaries, and we do not specify a subnet mask.

To configure Router B:

Router(config)# *router rip*
Router(config-router)# *network 172.17.0.0*
Router(config-router)# *network 172.18.0.0*

The routing table on Router A will look like:

RouterA# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:00, Serial0
```

The routing table on Router B will look like:

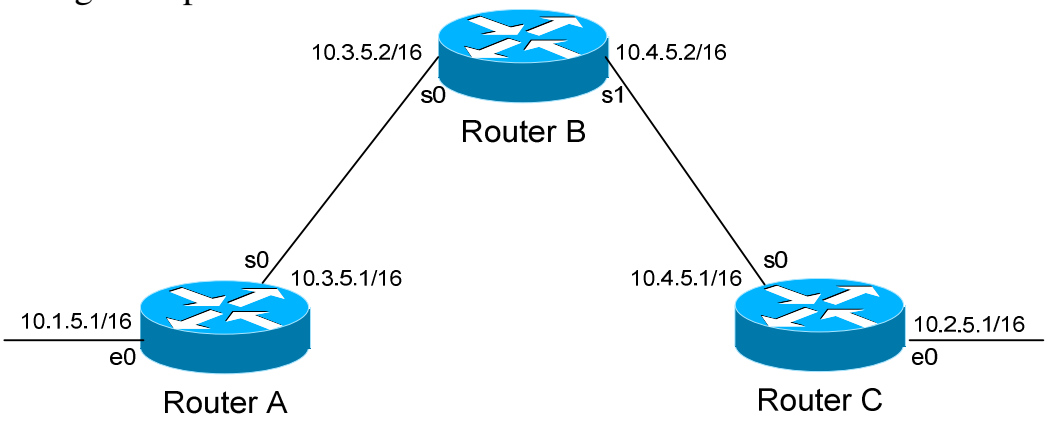
RouterB# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.17.0.0 is directly connected, Serial0
C     172.18.0.0 is directly connected, Ethernet0
R     172.16.0.0 [120/1] via 172.17.1.1, 00:00:00, Serial0
```

* * *

Limitations of RIPv1

The example on the previous page works fine with RIPv1, because the networks are contiguous and the subnet masks are consistent. Consider the following example:



This particular scenario will *still* work when using RIPv1, despite the fact that we've subnetted the major 10.0.0.0 network. Notice that the subnets are contiguous (that is, they belong to the same major network), and use the same subnet mask.

When Router A sends a RIPv1 update to Router B via Serial0, it will not include the subnet mask for the 10.1.0.0 network. However, because the 10.3.0.0 network is in the same major network as the 10.1.0.0 network, it will **not summarize** the address. The route entry in the update will simply state "10.1.0.0".

Router B will accept this routing update, and realize that the interface receiving the update (Serial0) belongs to the same major network as the route entry of 10.1.0.0. It will then apply the subnet mask of its Serial0 interface to this route entry.

Router C will similarly send an entry for the 10.2.0.0 network to Router B. Router B's routing table will thus look like:

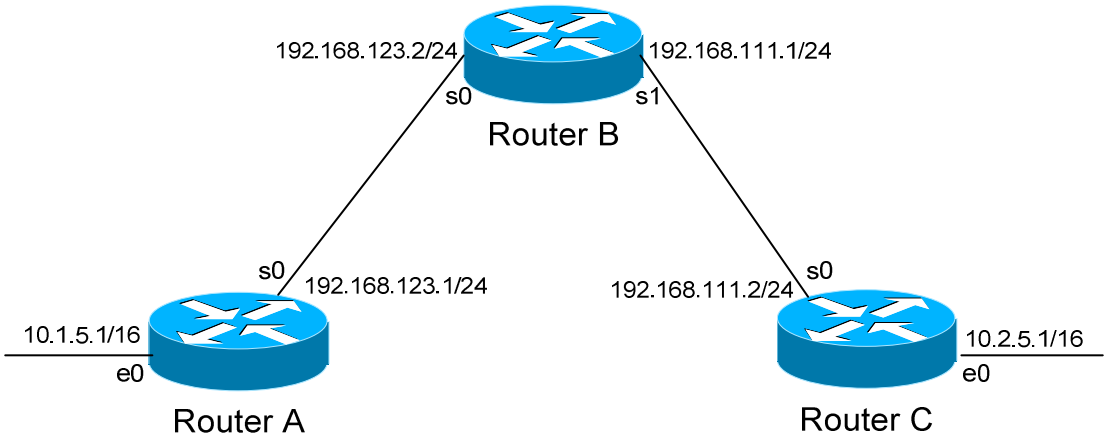
RouterB# *show ip route*

```
Gateway of last resort is not set

       10.0.0.0/16 is subnetted, 4 subnets
C      10.3.0.0 is directly connected, Serial0
C      10.4.0.0 is directly connected, Serial1
R      10.1.0.0 [120/1] via 10.3.5.1, 00:00:00, Serial0
R      10.2.0.0 [120/1] via 10.4.5.1, 00:00:00, Serial1
```

* * *

Limitations of RIPv1 (continued)

Consider the following, slightly altered, example:



We'll assume that RIPv1 is configured correctly on all routers. Notice that our networks are no longer contiguous. Both Router A and Router C contain *subnets* of the 10.0.0.0 major network (10.1.0.0 and 10.2.0.0 respectively).

Separating these networks now are two Class C subnets (192.168.123.0 and 192.168.111.0).

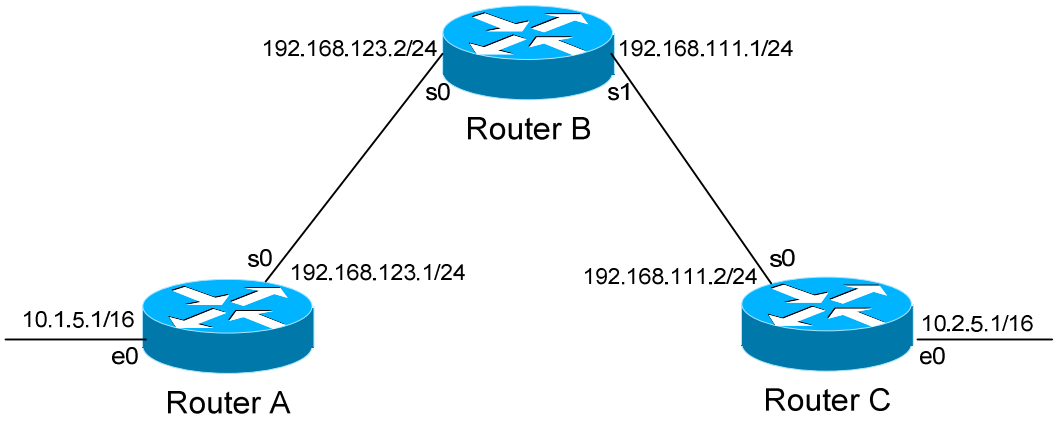
Why is this a problem? Again, when Router A sends a RIPv1 update to Router B via Serial, it will not include the subnet mask for the 10.1.0.0 network. Instead, Router A will consider itself a **border** router, as the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Router A will **summarize** the 10.1.0.0/16 network to its classful boundary of 10.0.0.0/8.

Router B will accept this routing update, and realize that it does not have a directly connected interface in the 10.x.x.x scheme. Thus, it has no subnet mask to apply to this route. Because of this, Router B will install the summarized 10.0.0.0 route into its routing table.

Router C, similarly, will consider itself a border router between networks 10.2.0.0 and 192.168.111.0. Thus, Router C will *also* send a summarized 10.0.0.0 route to Router B.

* * *

Limitations of RIPv1 (continued)



Router B's routing table will then look like:

RouterB# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
C     192.168.111.0 is directly connected, Serial1
R     10.0.0.0 [120/1] via 192.168.123.1, 00:00:00, Serial0
               [120/1] via 192.168.111.2, 00:00:00, Serial1
```

That's right, Router B now has two *equal* metric routes to get to the summarized 10.0.0.0 network, one through Router A and the other through Router C. Router B will now *load balance* all traffic to *any* 10.x.x.x network between routers A and C. Suffice to say, this is not a good thing. ☺

It gets better. Router B then tries to send routing updates to Router A and Router C, including the summary route of 10.0.0.0/8. Router A's routing table looks like:

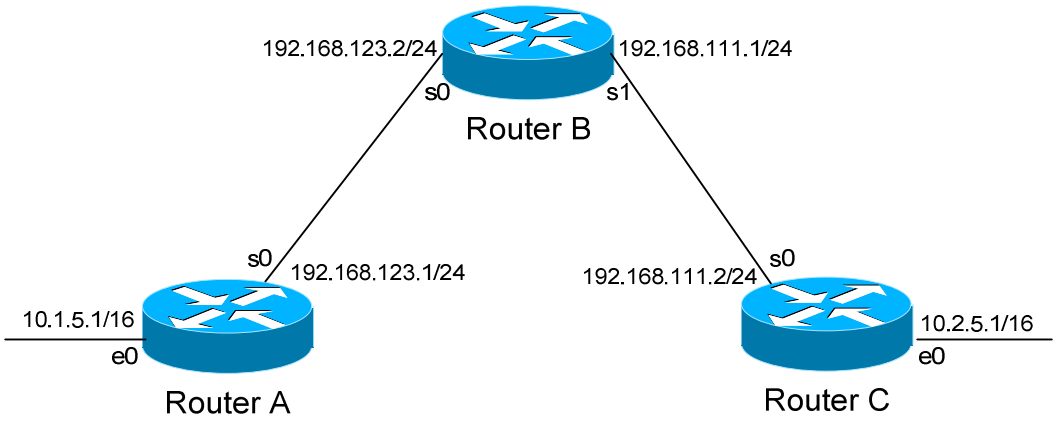
RouterA# *show ip route*

```
Gateway of last resort is not set

C     192.168.123.0 is directly connected, Serial0
       10.0.0.0/16 is subnetted, 1 subnet
C     10.1.0.0 is directly connected, Ethernet0
```

Router A will receive the summarized 10.0.0.0/8 route from Router B, and will reject it. This is because it already has the summary network of 10.0.0.0 in its routing table, and it's directly connected. Router C will respond exactly the same, and the 10.1.0.0/16 and 10.2.0.0/16 networks will *never* be able to communicate.

* * *

RIPv2 Configuration



RIPv2 overcomes the limitations of RIPv1 by including the subnet mask in its routing updates. By default, Cisco routers will use RIPv1. To change to Version 2, you must type:

Router(config)# *router rip*
Router(config-router)# *version 2*

Thus, the configuration of Router A would be:

RouterA(config)# *router rip*
RouterA(config-router)# *version 2*
RouterA(config-router)# *network 10.0.0.0*
RouterA(config-router)# *network 192.168.123.0*

Despite the fact that RIPv2 is a classless routing protocol, we still specify networks at their classful boundaries, without a subnet mask.

However, when Router A sends a **RIPv2** update to Router B via Serial0, by default it will still **summarize** the 10.1.0.0/16 network to 10.0.0.0/8. Again, this is because the 10.1.0.0 and 192.168.123.0 networks *do not* belong to the same major network. Thus, RIPv2 acts like RIPv1 in this circumstance…

…unless you disable **auto summarization:**

RouterA(config)# *router rip*
RouterA(config-router)# *version 2*
RouterA(config-router)# *no auto-summary*

The *no auto-summary* command will prevent Router A from summarizing the 10.1.0.0 network. Instead, Router A will send an update that includes both the subnetted network (10.1.0.0) and its subnet mask (255.255.0.0).

* * *

RIP Timers

RIP has four basic timers:

Update Timer (default **30 seconds**) – indicates how often the router will send out a routing table update.

Invalid Timer (default **180 seconds**) – indicates how long a route will remain in a routing table before being marked as invalid, if no new updates are heard about this route. The invalid timer will be reset if an update is received for that particular route *before* the timer expires.

A route marked as invalid is *not* immediately removed from the routing table. Instead, the route is marked (and advertised) with a metric of 16, indicating it is unreachable, and placed in a **hold-down** state.

Hold-down Timer (default **180 seconds**) – indicates how long RIP will "suppress" a route that it has placed in a **hold-down** state. RIP will not accept any new updates for routes in a hold-down state, until the hold-down timer expires.

A route will enter a hold-down state for one of three reasons:

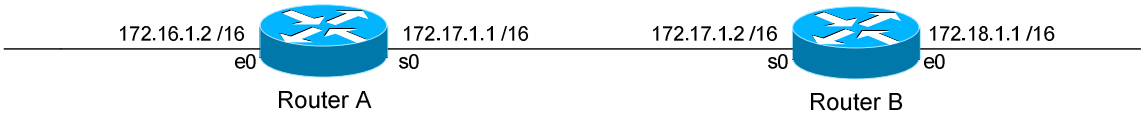
- The invalid timer has expired.
- An update has been received from another router, marking that route with a metric of 16 (or *unreachable*).
- An update has been received from another router, marking that route with a *higher* metric than what is currently in the routing table. This is to prevent loops.

Flush Timer (default **240 seconds**) **–** indicates how long a route can remain in a routing table before being flushed, if no new updates are heard about this route. The flush timer runs **concurrently with the invalid timer**, and thus will flush out a route 60 seconds after it has been marked invalid.

RIP timers must be identical on **all** routers on the RIP network, otherwise massive instability will occur.

* * *

RIP Timers Configuration and Example



Consider the above example. Router A receives a RIP update from Router B that includes network 172.18.0.0. Router A adds this network to its routing table:

RouterA# *show ip route*

```
Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:00, Serial0
```

Immediately, Router A sets an **invalid** timer of 180 seconds and **flush** timer of 240 seconds to this route, which run concurrently. If no update for this route is heard for 180 seconds, several things will occur:

- The route is marked as invalid in the routing table.
- The route enters a **hold-down** state (triggering the hold-down timer).
- The route is advertised to all other routers as unreachable.

The hold-down timer runs for 180 seconds *after* the route is marked as invalid. The router will not accept any new updates for this route until this hold-down period expires.

If no update is heard *at all*, the route will be removed from the routing table once the flush timer expires, which is 60 seconds after the route is marked as invalid. Remember that the invalid and flush timers run concurrently.

To configure the RIP timers:

Router(config)# *router rip*
Router(config-router)# *timers basic 20 120 120 160*

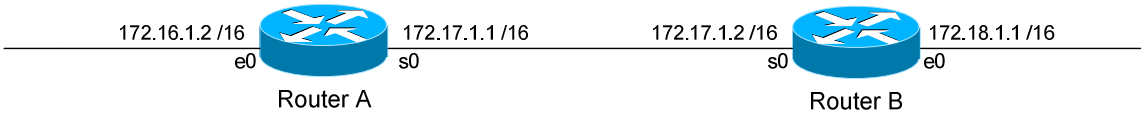
The *timers basic* command allows us to change the update (*20*), invalid (*120*), hold-down (*120)*, and flush *(240)* timers. To return the timers back to their defaults:

Router(config-router)# *no timers basic*

* * *

RIP Loop Avoidance Mechanisms

RIP, as a Distance Vector routing protocol, is susceptible to loops.



Let's assume no loop avoidance mechanisms are configured on either router. If the 172.18.0.0 network fails, Router B will send out an update to Router A within 30 seconds (whenever its update timer expires) stating that route is unreachable (metric = 16).

But what if an update from Router A reaches Router B *before* this can happen? Router A believes it can reach the 172.18.0.0 network in one hop (through Router B). This will cause Router B to believe it can reach the failed 172.18.0.0 network in **two hops**, through Router A. Both routers will continue to increment the metric for the network until they reach a hop count of **16**, which is unreachable. This behavior is known as **counting to infinity**.

How can we prevent this from happening? There are several loop avoidance mechanisms:

Split-Horizon – Prevents a routing update from being sent out the interface it was received on. In our above example, this would prevent Router A from sending an update for the 172.18.0.0 network *back* to Router B, as it originally learned the route from Router B. Split-horizon is **enabled** by default on Cisco Routers.

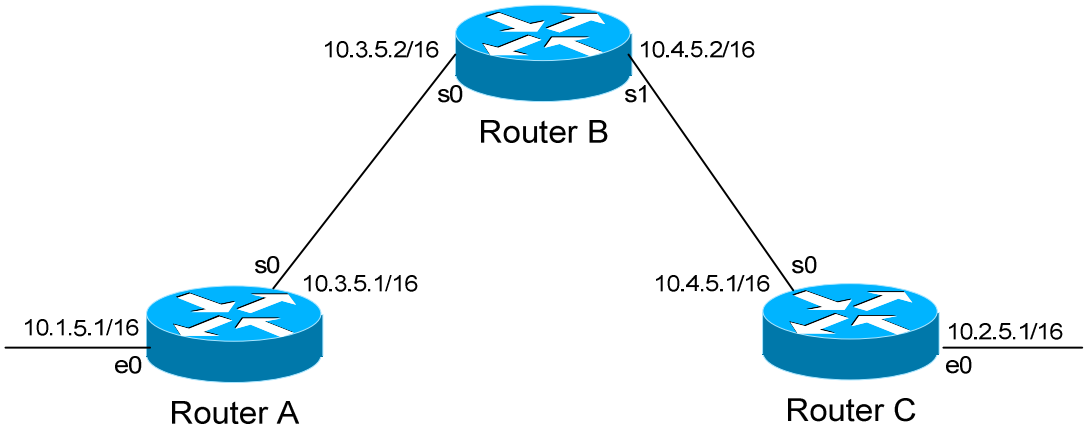
Route-Poisoning – Works in conjunction with split-horizon, by **triggering** an automatic update for the failed network, without waiting for the update timer to expire. This update is sent out all interfaces with an infinity metric for that network.

Hold-Down Timers – Prevents RIP from accepting any new updates for routes in a hold-down state, until the hold-down timer expires. If Router A sends an update to Router B with a *higher* metric than what is currently in Router B's routing table, that route will be placed in a hold-down state. (Router A's metric for the 172.18.0.0 network is 1; while Router B's metric is 0).

* * *

RIP Passive Interfaces

It is possible to control which router interfaces will participate in the RIP process.



Consider the following scenario. Router C does not want to participate in the RIP domain. However, it still wants to *listen* to updates being sent from Router B, just not *send* any updates back to Router B:

RouterC(config)# *router rip*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface s0*

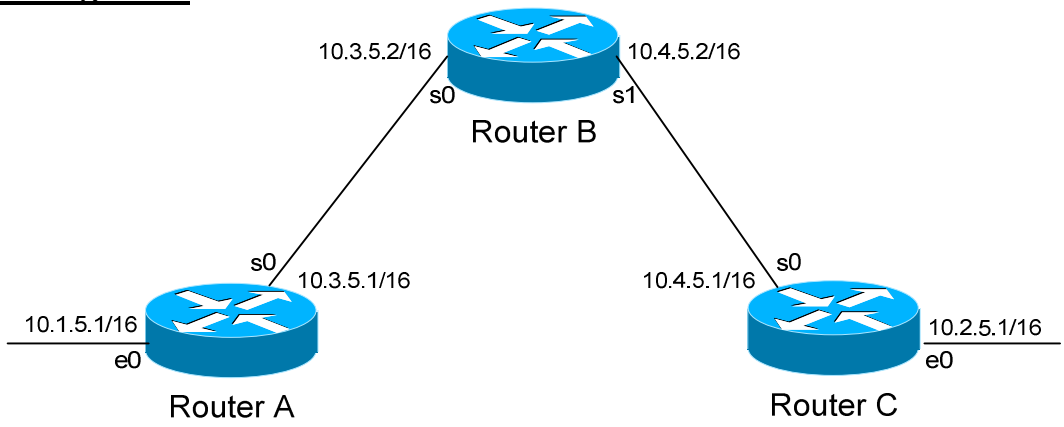
The *passive-interface* command will prevent updates from being **sent** out of the Serial0 interface, but Router C will still **receive** updates on this interface.

We can configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces we **do** want updates to be sent out:

RouterC(config)# *router rip*
RouterC(config-router)# *network 10.4.0.0*
RouterC(config-router)# *network 10.2.0.0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

* * *

RIP Neighbors



Recall that RIPv1 sends out its updates as **broadcasts**, whereas RIPv2 sends out its updates as **multicasts** to the 224.0.0.9 address. We can configure specific RIP *neighbor* commands, which will allow us to **unicast** routing updates to those neighbors.

On Router B:

RouterB(config)# *router rip*
RouterB(config-router)# *network 10.3.0.0*
RouterB(config-router)# *network 10.4.0.0*
RouterB(config-router)# *neighbor 10.3.5.1*
RouterB(config-router)# *neighbor 10.4.5.1*

Router B will now unicast RIP updates to Router A and Router C.

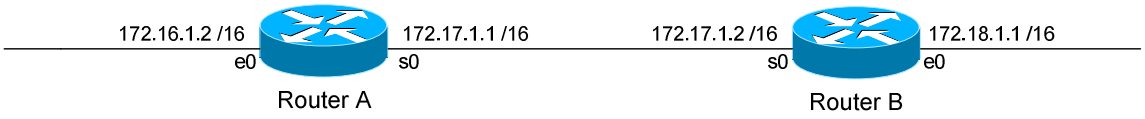
However, Router B will still broadcast (if RIPv1) or multicast (if RIPv2) its updates, in addition to sending unicast updates to its neighbors. In order to prevent broadcast/multicast updates, we must also use **passive interfaces**:

RouterB(config)# *router rip*
RouterB(config-router)# *passive-interface s0*
RouterB(config-router)# *passive-interface s1*
RouterB(config-router)# *neighbor 10.3.5.1*
RouterB(config-router)# *neighbor 10.4.5.1*

The *passive-interface* commands prevent the updates from being broadcasted or multicasted. The *neighbor* commands still allow unicast updates to those specific neighbors.

* * *

RIPv2 Authentication



RIPv2 supports authentication to secure routing updates.

The first step is creating a shared authentication *key* that must be identical on both routers. This is accomplished in global configuration mode:

RouterA(config)# *key chain MYCHAIN*
RouterA(config-keychain)# *key 1*
RouterA(config-keychain-key)# *key-string MYPASSWORD*

RouterB(config)# *key chain MYCHAIN*
RouterB(config-keychain)# *key 1*
RouterB(config-keychain-key)# *key-string MYPASSWORD*

The first command creates a *key chain* called *MYCHAIN*. We must then associate a *key* to our keychain. Then we actually configure the shared key using the *key-string* command.

We then apply our key chain to the interface connecting to the other router:

RouterA(config)# *interface s0*
RouterA(config-if)# *ip rip authentication key-chain MYCHAIN*

RouterB(config)# *interface s0*
RouterB(config-if)# *ip rip authentication key-chain MYCHAIN*

If there was another router off of Router B's Ethernet port, we could create a *separate* key chain with a different key-string. Every router on the RIP domain does not need to use the same key chain, only interfaces directly connecting two (or more) routers.

The final step in configuring authentication is identifying which encryption to use. By default, the key is sent in clear text:

RouterA(config)# *interface s0*
RouterA(config-if)# *ip rip authentication mode text*

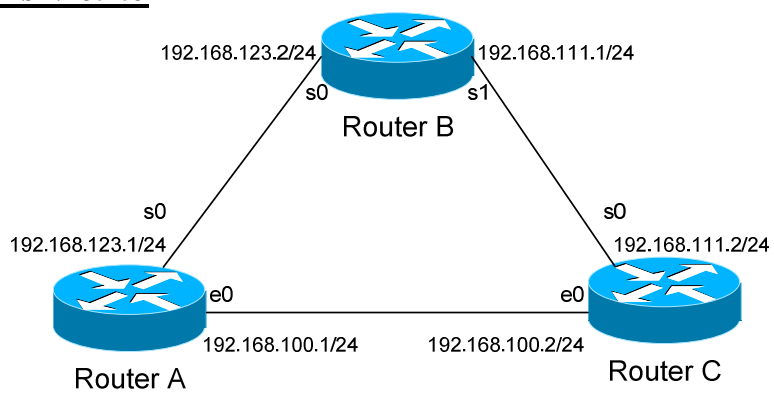
Or we can use MD5 encryption for additional security:

RouterA(config)# *interface s0*
RouterA(config-if)# *ip rip authentication mode md5*

Whether text or MD5 is used, it **must** be the same on both routers.

* * *

Altering RIP's Metric



Consider the above example. Router B has two paths to get to the 192.168.100.0 network, via Router A and Router C. Because the metric is equal (1 hop), Router B will load balance between these two paths.

What if we wanted Router B to only go through Router A, and use Router C only as a backup? To accomplish this, we can adjust RIP's metric to make one route more preferred than the other.

The first step is creating an access-list on Router B that defines which route we wish to alter:

RouterB(config)# *ip access-list standard MYLIST*
RouterB(config-std-nacl)# *permit 192.168.100.0 0.0.0.255*

Next, we tell RIP how much to **offset** this route if received by Router C:

RouterB(config)# *router rip*
RouterB(config-router)# *offset-list MYLIST in 4 s1*

We specify an *offset-list* pointing to our access list named *MYLIST*. We will increase the routing metric by *4* for that route coming *in*bound to interface *Serial 1*.

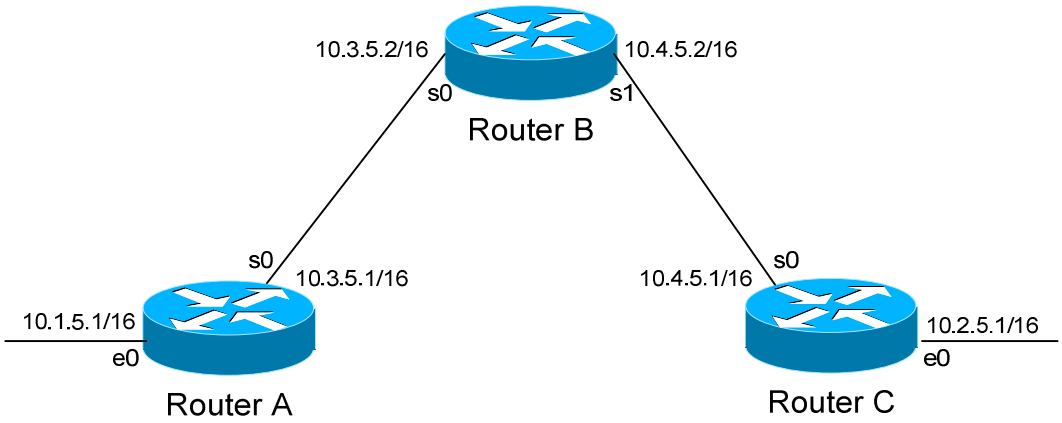
Thus, when Router C sends an update to Router C for the 192.168.100.0 network, Router B will increase its metric of 1 hop to 5 hops, thus making Router A's route preferred.

We could have also configured Router C to **advertise** that route with a higher metric (notice the *out* in the *offset-list* command):

RouterC(config)# *ip access-list standard MYLIST*
RouterC(config-std-nacl)# *permit 192.168.100.0 0.0.0.255*
RouterC(config)# *router rip*
RouterC(config-router)# *offset-list MYLIST out 4 s0*

* * *

Interoperating between RIPv1 and RIPv2



Recall that, with some configuration, RIPv1 and RIPv2 can interoperate. By default:

- RIPv1 routers will sent only Version 1 packets
- RIPv1 routers will receive both Version 1 and 2 updates
- RIPv2 routers will both send and receive only Version 2 updates

If Router A is running RIP v1, and Router B is running RIP v2, some additional configuration is necessary.

Either we must configure Router A to send Version 2 updates:

RouterA(config)# *interface s0*
RouterA(config-if)# *ip rip send version 2*

Or configure Router B to accept Version 1 updates.

RouterB(config)# *interface s0*
RouterB(config-if)# *ip rip receive version 1*

Notice that this is configured on an interface. Essentially, we're configuring the version of RIP on a per-interface basis.

We can also have an interface send or receive both versions simultaneously:

RouterB(config)# *interface s0*
RouterB(config-if)# *ip rip receive version 1 2*

We can further for RIPv2 to send broadcast updates, instead of multicasts:

RouterB(config)# *interface s0*
RouterB(config)# *ip rip v2-broadcast*

* * *

Triggering RIP Updates

On point-to-point interfaces, we can actually force RIP to only send routing updates if there is a change:

RouterB(config)# *interface s0.150 point-to-point*
RouterB(config-if)# *ip rip triggered*

Again, this is only applicable to **point-to-point** links. We cannot configure RIP triggered updates on an Ethernet network.

Troubleshooting RIP

Various troubleshooting commands exist for RIP.

To view the IP routing table:

Router# *show ip route*

```
<eliminated irrelevant header>

Gateway of last resort is not set

C     172.16.0.0 is directly connected, Ethernet0
C     172.17.0.0 is directly connected, Serial0
R     172.18.0.0 [120/1] via 172.17.1.2, 00:00:15, Serial0
R     192.168.123.0 [120/1] via 172.16.1.1, 00:00:00, Ethernet0
```

To view a specific route within the IP routing table:

Router# *show ip route 172.18.0.0*

```
Routing entry for 172.18.0.0/16
      Known via "rip", distance 120, metric 1
      Last update from 172.17.1.2 on Serial 0, 00:00:15 ago
```

To debug RIP in real time:

Router# *debug ip rip*

* * *

Troubleshooting RIP (continued)

To view information specific to the RIP protocol:

Router# *show ip protocols*

```
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 20 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Incoming routes will have 4 added to metric if on list 1
  Redistributing: connected, static, rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Serial0               1 2   1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.16.0.0
    172.17.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.17.1.2           120      00:00:17
  Distance: (default is 120)
```

This command provides us with information on RIP timers, on the RIP versions configured on each interface, and the specific networks RIP is advertising.

To view *all* routes in the RIP database, and not just the entries added to the routing table:

Router# *show ip rip database*

```
7.0.0.0/8    auto-summary
7.0.0.0/8
    [5] via 172.16.1.1, 00:00:06, Ethernet0
172.16.0.0/16    directly connected, Ethernet0
172.17.0.0/16    directly connected, Serial0
```

* * *

- Open Shortest Path First -

OSPF (Open Shortest Path First)

OSPF is a standardized Link-State routing protocol, designed to scale efficiently to support larger networks.

OSPF adheres to the following Link State characteristics:

- OSPF employs a hierarchical network design using **Areas.**
- OSPF will form **neighbor** relationships with adjacent routers in the same **Area.**
- Instead of advertising the *distance* to connected networks, OSPF advertises the *status* of directly connected **links** using **Link-State Advertisements (LSAs)**.
- OSPF sends updates (LSAs) when there is a change to one of its links, and will *only* send the change in the update. LSAs are additionally refreshed every **30 minutes**.
- OSPF traffic is multicast either to address **224.0.0.5 (**all OSPF routers) or **224.0.0.6** (all Designated Routers).
- OSPF uses the **Dijkstra Shortest Path First** algorithm to determine the shortest path.
- OSPF is a classless protocol, and thus supports VLSMs.

Other characteristics of OSPF include:

- OSPF supports only IP routing.
- OSPF routes have an administrative distance is **110**.
- OSPF uses **cost** as its metric, which is computed based on the bandwidth of the link. OSPF has no hop-count limit.

The OSPF process builds and maintains three separate tables:

- A **neighbor table** – contains a list of all neighboring routers.
- A **topology table** – contains a list of *all* possible routes to all known networks within an area.
- A **routing table** – contains the *best* route for each known network.

* * *

OSPF Neighbors

OSPF forms neighbor relationships, called **adjacencies,** with other routers in the same **Area** by exchanging **Hello** packets to multicast address **224.0.0.5**. Only after an adjacency is formed can routers share routing information.

Each OSPF router is identified by a unique **Router ID.** The Router ID can be determined in one of three ways:

- The Router ID can be **manually** specified.
- If not manually specified, the highest IP address configured on any **Loopback interface** on the router will become the Router ID.
- If no loopback interface exists, the highest IP address configured on any **Physical interface** will become the Router ID.

By default, Hello packets are sent out OSPF-enabled interfaces every **10 seconds** for broadcast and point-to-point interfaces, and **30 seconds** for non-broadcast and point-to-multipoint interfaces.

OSPF also has a **Dead Interval**, which indicates how long a router will wait without hearing any hellos before announcing a neighbor as "down." Default for the Dead Interval is **40 seconds** for broadcast and point-to-point interfaces, and **120** seconds for non-broadcast and point-to-multipoint interfaces. Notice that, by default, the dead interval timer is four times the Hello interval.

These timers can be adjusted on a *per interface* basis:

Router(config-if)# *ip ospf hello-interval 15*
Router(config-if)# *ip ospf dead-interval 60*

* * *

OSPF Neighbors (continued)

OSPF routers will only become neighbors if the following parameters within a Hello packet are identical on each router:

- Area ID
- Area Type (stub, NSSA, etc.)
- Prefix
- Subnet Mask
- Hello Interval
- Dead Interval
- Network Type (broadcast, point-to-point, etc.)
- Authentication

The Hello packets also serve as **keepalives** to allow routers to quickly discover if a neighbor is down. Hello packets also contain a **neighbor field** that lists the Router IDs of all neighbors the router is connected to.

A **neighbor table** is constructed from the OSPF Hello packets, which includes the following information:

- The **Router ID** of each neighboring router
- The current "state" of each neighboring router
- The interface directly connecting to each neighbor
- The IP address of the remote interface of each neighbor

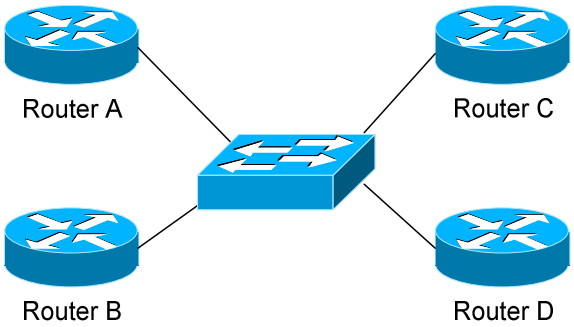
(Reference: http://www.cisco.com/warp/public/104/29.html)

* * *

OSPF Designated Routers

In multi-access networks such as Ethernet, there is the possibility of *many* neighbor relationships on the same physical segment. In the above example, four routers are connected into the same multi-access segment. Using the following formula (where "n" is the number of routers):

n(n-1)/2



…..it is apparent that **6** separate adjacencies are needed for a fully meshed network. Increase the number of routers to five, and **10** separate adjacencies would be required. This leads to a considerable amount of unnecessary Link State Advertisement (LSA) traffic.

If a link off of Router A were to fail, it would flood this information to all neighbors. Each neighbor, in turn, would then flood that same information to all other neighbors. This is a waste of bandwidth and processor load.

To prevent this, OSPF will elect a **Designated Router (DR)** for each multi-access networks, accessed via multicast address **224.0.0.6**. For redundancy purposes, a **Backup Designated Router (BDR)** is also elected.

OSPF routers will form adjacencies with the DR and BDR. If a change occurs to a link, the update is forwarded only to the DR, which then forwards it to all other routers. This greatly reduces the flooding of LSAs.

DR and BDR elections are determined by a router's **OSPF priority**, which is configured on a per-interface basis (a router can have interfaces in multiple multi-access networks). The router with the **highest priority** becomes the DR; second highest becomes the BDR. If there is a tie in priority, whichever router has the **highest Router ID** will become the DR. To change the priority on an interface:

Router(config-if)# *ip ospf priority 125*

Default priority on Cisco routers is **1**. A priority of **0** will prevent the router from being elected DR or BDR. **Note:** The DR election process is ***not preemptive***. Thus, if a router with a higher priority is added to the network, it will *not* automatically supplant an existing DR. Thus, a router that should never become the DR should always have its priority set to 0.

* * *

OSPF Neighbor States

Neighbor adjacencies will progress through several **states**, including:

Down – indicates that no Hellos have been heard from the neighboring router.

Init – indicates a Hello packet has been heard from the neighbor, but two-way communication has not yet been initialized.

2-Way – indicates that bidirectional communication has been established. Recall that Hello packets contain a *neighbor* field. Thus, communication is considered 2-Way once a router sees its own Router ID in its neighbor's Hello Packet. **Designated** and **Backup Designated Routers** are elected at this stage.

ExStart – indicates that the routers are preparing to share link state information. Master/slave relationships are formed between routers to determine who will begin the exchange.

Exchange – indicates that the routers are exchanging **Database Descriptors (DBDs).** DBDs contain a description of the router's Topology Database. A router will examine a neighbor's DBD to determine if it has information to share.

Loading – indicates the routers are finally exchanging **Link State Advertisements,** containing information about all links connected to each router. Essentially, routers are sharing their topology tables with each other.

Full – indicates that the routers are fully synchronized. The topology table of all routers in the area should now be identical. Depending on the "role" of the neighbor, the state may appear as:

- **Full/DR** – indicating that the neighbor is a Designated Router (DR)
- **Full/BDR** – indicating that the neighbor is a Backup Designated Router (BDR)
- **Full/DROther –** indicating that the neighbor is neither the DR or BDR

On a multi-access network, OSPF routers will *only* form Full adjacencies with DRs and BDRs. Non-DRs and non-BDRs will still form adjacencies, but will remain in a **2-Way State**. This is normal OSPF behavior.

* * *

OSPF Network Types

OSPF's functionality is different across several different network topology types. OSPF's interaction with Frame Relay will be explained in another section

Broadcast Multi-Access – indicates a topology where broadcast occurs.

- Examples include Ethernet, Token Ring, and ATM.
- OSPF *will* elect DRs and BDRs.
- Traffic to DRs and BDRs is multicast to 224.0.0.6. Traffic from DRs and BDRs to other routers is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Point-to-Point – indicates a topology where two routers are directly connected.

- An example would be a point-to-point T1.
- OSPF *will not* elect DRs and BDRs.
- All OSPF traffic is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Point-to-Multipoint – indicates a topology where one interface can connect to multiple destinations. Each connection between a source and destination is treated as a point-to-point link.

- An example would be Point-to-Multipoint Frame Relay.
- OSPF *will not* elect DRs and BDRs.
- All OSPF traffic is multicast to 224.0.0.5.
- Neighbors *do not* need to be manually specified.

Non-broadcast Multi-access Network (NBMA) – indicates a topology where one interface can connect to multiple destinations; however, broadcasts cannot be sent across a NBMA network.

- An example would be Frame Relay.
- OSPF *will* elect DRs and BDRs.
- OSPF neighbors must be *manually* defined, thus All OSPF traffic is unicast instead of multicast.

Remember: on *non-broadcast* networks, neighbors must be **manually specified**, as multicast Hello's are not allowed.

* * *

Configuring OSPF Network Types

The default OSPF network type for basic Frame Relay is **Non-broadcast Multi-access Network (NBMA).** To configure manually:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay map ip 10.1.1.1 101
Router(config-if)# ip ospf network non-broadcast

Router(config)# router ospf 1
Router(config-router)# neighbor 10.1.1.1
```

Notice that the *neighbor* was manually specified, as multicasting is not allowed on an NBMA. However, the Frame-Relay network can be tricked into allowing broadcasts, eliminating the need to manually specify neighbors:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay
Router(config-if)# frame-relay map ip 10.1.1.1 101 broadcast
Router(config-if)# ip ospf network broadcast
```

Notice that the *ospf network* type has been changed to *broadcast,* and the *broadcast* parameter was added to the *frame-relay map* command. The neighbor no longer needs to be specified, as multicasts will be allowed out this map.

The default OSPF network type for Ethernet and Token Ring is **Broadcast Multi-Access.** To configure manually:

```
Router(config)# interface e0
Router(config-if)# ip ospf network broadcast
```

The default OSPF network type for T1's (HDLC or PPP) and Point-to-Point Frame Relay is **Point-to-Point.** To configure manually:

```
Router(config)# interface s0
Router(config-if)# encapsulation frame-relay

Router(config)# interface s0.1 point-to-point
Router(config-if)# frame-relay map ip 10.1.1.1 101 broadcast
Router(config-if)# ip ospf network point-to-point
```

* * *

Configuring OSPF Network Types (continued)

The default OSPF network type for Point-to-Multipoint Frame Relay is *still* **Non-broadcast Multi-access Network (NBMA).** However, OSPF supports an additional network type called **Point-to-Multipoint,** which will allow neighbor discovery to occur automatically**.** To configure:

Router(config)# *interface s0*
Router(config-if)# *encapsulation frame-relay*

Router(config)# *interface s0.2 multipoint*
Router(config-if)# *frame-relay map ip 10.1.1.1 101 broadcast*
Router(config-if)# *ip ospf network point-to-multipoint*

Additionally, a *non-broadcast* parameter can be added to the *ip ospf network* command when specifying *point-to-multipoint.*

Router(config)# *interface s0*
Router(config-if)# *encapsulation frame-relay*

Router(config)# *interface s0.2 multipoint*
Router(config-if)# *frame-relay map ip 10.1.1.1 101*
Router(config-if)# *ip ospf network point-to-multipoint non-broadcast*

Router(config)# *router ospf 1*
Router(config-router)# *neighbor 10.1.1.1*

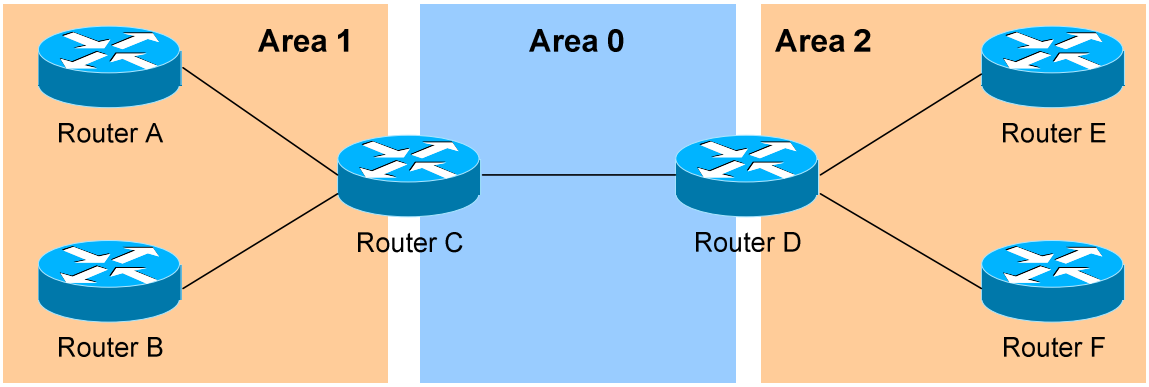
Notice the different in configuration. The *frame-relay map* command no longer has the *broadcast* parameter, as broadcasts and multicasts are not allowed on a non-broadcast network.

Thus, in the OSPF router configuration, neighbors must again be manually specified. Traffic to those neighbors will be **unicast** instead of multicast.

OSPF network types must be set identically on two "neighboring" routers, otherwise they will never form an adjacency.

* * *

The OSPF Hierarchy



OSPF is a hierarchical system that separates an Autonomous System into individual **areas**. OSPF traffic can either be **intra-area** (within one area), **inter-area** (between separate areas), or **external** (from another AS).

OSPF routers build a **Topology Database** of all **links** within their area, and all routers within an area will have an *identical* topology database. Routing updates between these routers will *only* contain information about links local to their area. Limiting the topology database to include only the local area conserves bandwidth and reduces CPU loads.

Area 0 is required for OSPF to function, and is considered the "**Backbone**" area. As a rule, all other areas must have a connection into Area 0, though this rule can be bypassed using **virtual links** (explained shortly). Area 0 is often referred to as the *transit* area to connect all other areas.

OSPF routers can belong to multiple areas, and will thus contain separate Topology databases for each area. These routers are known as **Area Border Routers (ABRs)**.

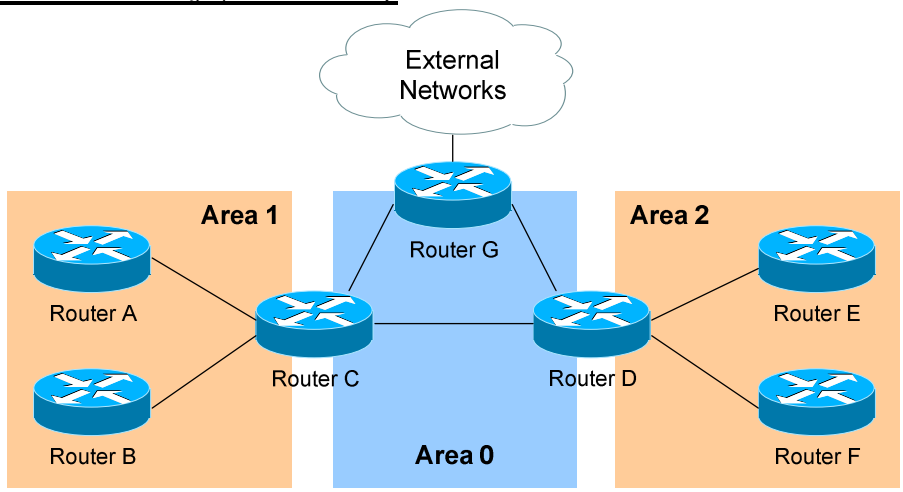
Consider the above example. Three areas exist: Area 0, Area 1, and Area 2. Area 0, again, is the backbone area for this Autonomous System. Both Area 1 and Area 2 must directly connect to Area 0.

Routers A and B belong fully to Area 1, while Routers E and F belong fully to Area 2. These are known as **Internal Routers**.

Router C belongs to both Area 0 and Area 1. Thus, it is an **ABR**. Because it has an interface in Area 0, it can also be considered a **Backbone Router**. The same can be said for Router D, as it belongs to both Area 0 and Area 2.

* * *

The OSPF Hierarchy (continued)



Now consider the above example. Router G has been added, which belongs to Area 0. However, Router G also has a connection to the Internet, which is outside this Autonomous System.

This makes Router G an **Autonomous System Border Router** (**ASBR**). A router can become an ASBR in one of two ways:

- By connecting to a separate Autonomous System, such as the Internet
- By redistributing another routing protocol into the OSPF process.

ASBRs provide access to *external* networks. OSPF defines two "types" of external routes:

- **Type 2 (E2)** – Includes only the external cost to the destination network. External cost is the metric being advertised from outside the OSPF domain. This is the default type assigned to external routes.
- **Type 1 (E1)** – Includes both the external cost, and the internal cost to reach the ASBR, to determine the total metric to reach the destination network. Type 1 routes are always *preferred* over Type 2 routes to the same destination.

Thus, the four separate OSPF router types are as follows:

- **Internal Routers –** all router interfaces belong to only one Area.
- **Area Border Routers (ABRs) –** contains interfaces in at least two separate areas
- **Backbone Routers –** contain at least one interface in Area 0
- **Autonomous System Border Routers (ASBRs) –** contain a connection to a separate Autonomous System

* * *

LSAs and the OSPF Topology Database

OSPF, as a link-state routing protocol, does not rely on *routing-by-rumor* as RIP and IGRP do.

Instead, OSPF routers keep track of the status of **links** within their respective areas. A link is simply a router interface. From these lists of links and their respective statuses, the topology database is created. OSPF routers forward **link-state advertisements (LSAs)** to ensure the topology database is consistent on each router within an area.

Several LSA types exist:

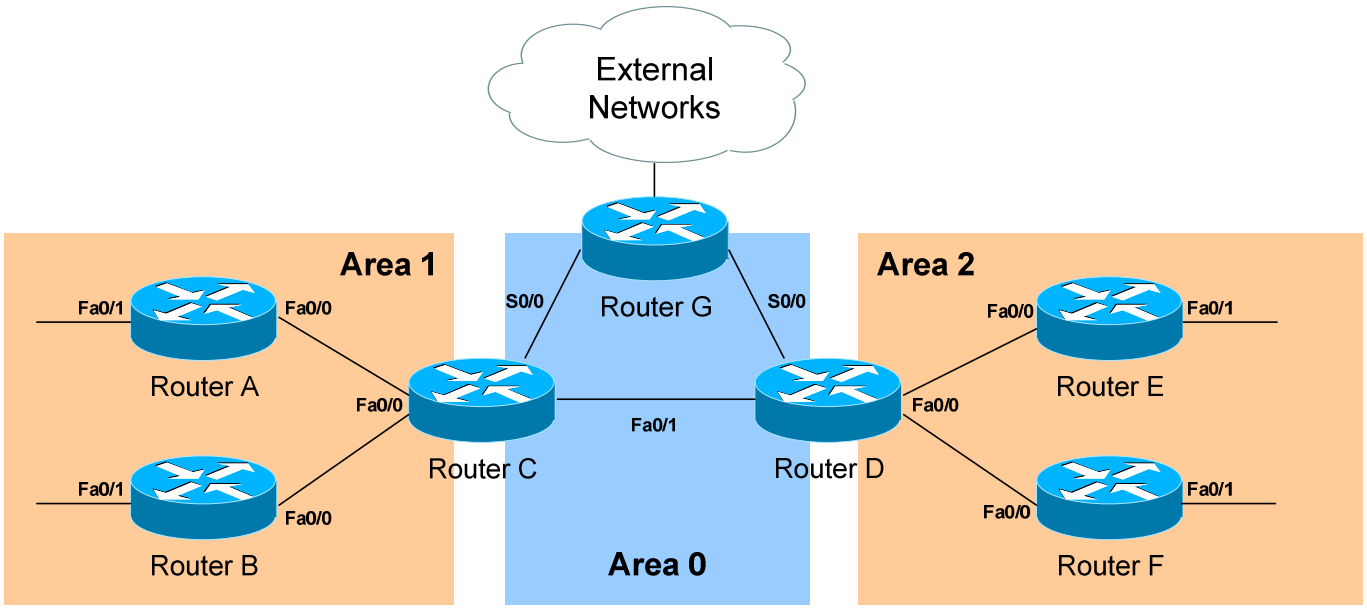
- **Router LSA (Type 1) –** Contains a list of all links local to the router, and the status and "cost" of those links. Type 1 LSAs are generated by all routers in OSPF, and are flooded to all other routers within the local area.
- **Network LSA (Type 2) –** Generated by all Designated Routers in OSPF, and contains a list of all routers attached to the Designated Router.
- **Network Summary LSA (Type 3)** – Generated by all ABRs in OSPF, and contains a list of all destination networks within an area. Type 3 LSAs are sent between areas to allow inter-area communication to occur.
- **ASBR Summary LSA (Type 4)** – Generated by ABRs in OSPF, and contains a *route* to any ASBRs in the OSPF system. Type 4 LSAs are sent from an ABR into its local area, so that Internal routers know how to exit the Autonomous System.
- **External LSA (Type 5)** – Generated by ASBRs in OSPF, and contain routes to destination networks *outside* the local Autonomous System. Type 5 LSAs can also take the form of a **default route** to all networks outside the local AS. Type 5 LSAs are flooded to all areas in the OSPF system.

Multicast OSPF (MOSPF) utilizes a Type 6 LSA, but that goes beyond the scope of this guide.

Later in this section, **Type 7 NSSA External LSAs** will be described in detail.

* * *

LSAs and the OSPF Topology Database (continued)



From the above example, the following can be determined:

- Routers A, B, E, and F are **Internal Routers.**
- Routers C and D are **ABRs.**
- Router G is an **ASBR.**

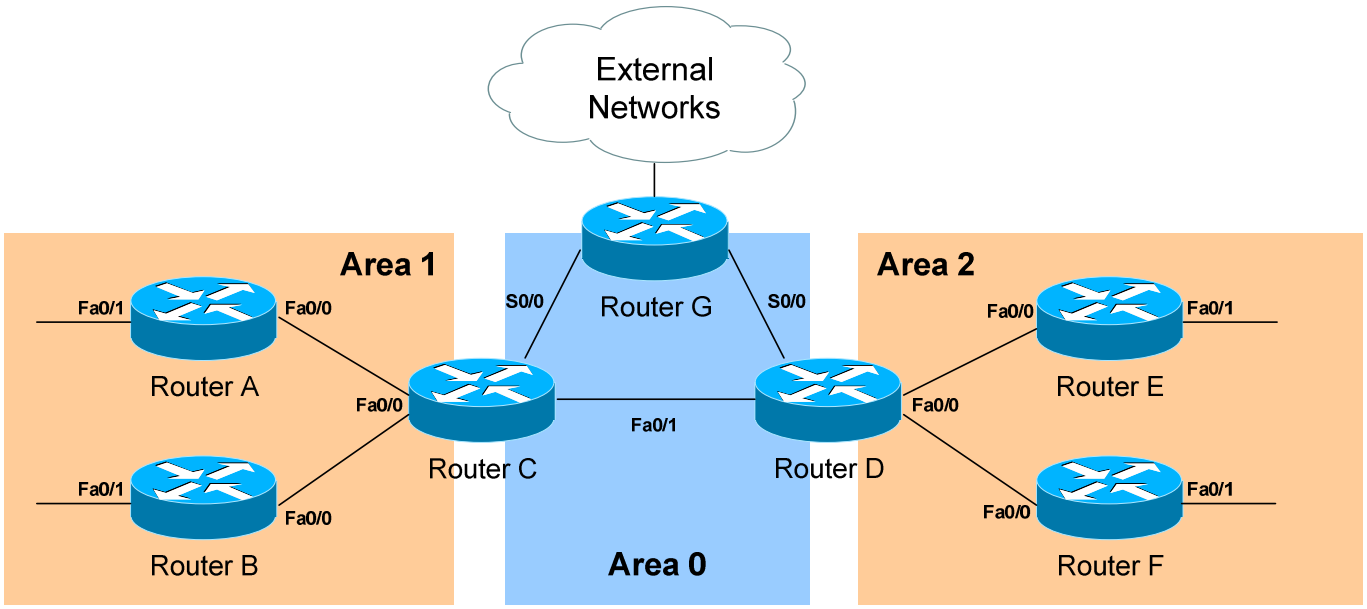
All routers will generate **Router** (**Type 1) LSAs**. For example, Router A will generate a Type 1 LSA that contains the status of links FastEthernet 0/0 and FastEthernet 0/1. This LSA will be flooded to all other routers in Area 1.

Designated Routers will generate **Network (Type 2) LSAs.** For example, if Router C was elected the DR for the multi-access network in Area 1, it would generate a Type 2 LSA containing a list of all routers attached to it.

Area Border Routers (ABRs) will generate **Network Summary (Type 3) LSAs**. For example, Router C is an ABR between Area 0 and Area 1. It will thus send Type 3 LSAs into *both* areas. Type 3 LSAs sent into Area 0 will contain a list of networks within Area 1, including **costs** to reach those networks. Type 3 LSAs sent into Area 1 will contain a list of networks within Area 0, *and* all other areas connected to Area 0. This allows Area 1 to reach any other area, and all other areas to reach Area 1.

* * *

LSAs and the OSPF Topology Database (continued)



ABRs will also generate **ASBR Summary (Type 4) LSAs.** For example, Router C will send Type 4 LSAs into Area 1 containing a route to the ASBR, thus providing routers in Area 1 with the path out of the Autonomous System.

ASBRs will generate **External (Type 5) LSAs.** For example, Router G will generate Type 5 LSAs that contain routes to network outside the AS. These Type 5 LSAs will be flooded to routers of all areas.

Each type of LSA is propagated under three circumstances:

- When a new adjacency is formed.
- When a change occurs to the topology table.
- When an LSA reaches its maximum age (every **30 minutes,** by default).

Thus, though OSPF is typically recognized to only send updates when a change occurs, LSA's are still periodically refreshed every 30 minutes.

* * *

The OSPF Metric

OSPF determines the best (or *shortest)* path to a destination network using a **cost** metric, which is based on the bandwidth of interfaces. The *total* cost of a route is the sum of all *outgoing* interface costs. Lowest cost is preferred.

Cisco applies default costs to specific interface types:

Type	***Cost***
Serial (56K)	*1785*
Serial (64K)	*1562*
T1 (1.544Mbps)	*64*
Token Ring (4Mbps)	*25*
Ethernet (10 Mbps)	*10*
Token Ring (16 Mbps)	*6*
Fast Ethernet	*1*

On Serial interfaces, OSPF will use the configured *bandwidth* (measured in Kbps) to determine the cost:

Router(config)# *interface s0*
Router(config-if)# *bandwidth 64*

The default cost of an interface can be superseded:

Router(config)# *interface e0*
Router(config-if)# *ip ospf cost 5*

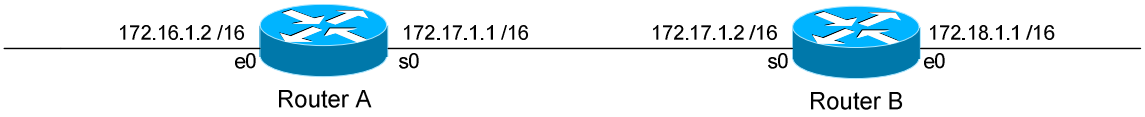
Changing the cost of an interface can alter which path OSPF deems the "shortest," and thus should be used with great care.

To alter how OSPF calculates its default metrics for interfaces:

Router(config)# *router ospf 1*
Router(config-router)# *ospf auto-cost reference-bandwidth 100*

The above *ospf auto-cost* command has a value of *100* configured, which is actually the default. This indicates that a 100Mbps link will have a cost of 1 (because 100/100 is 1). All other costs are based off of this. For example, the cost of 4 Mbps Token Ring is 25 because 100/4 = 25.

Configuring Basic OSPF



Routing protocol configuration occurs in Global Configuration mode. On Router A, to configure OSPF:

RouterA(config)# *router ospf 1*
RouterA(config-router)# *router-id 1.1.1.1*
RouterA(config-router)# *network 172.16.0.0 0.0.255.255 area 1*
RouterA(config-router)# *network 172.17.0.0 0.0.255.255 area 0*

The first command, *router ospf 1*, enables the OSPF process. The "*1*" indicates the OSPF process ID, and can be unique on each router. The process ID allows multiple OSPF processes to run on the same router. The *router-id* command assigns a unique OSPF ID of *1.1.1.1* for this router.

Note the use of a wildcard mask instead of a subnet mask in the *network* statement. With OSPF, we're *not* telling the router what networks to advertise; we're telling the router to place certain interfaces into specific areas, so those routers can form neighbor relationships. The wildcard mask *0.0.255.255* tells us that the last two octets can match any number.

The first *network* statement places interface E0 on Router A into Area 1. Likewise, the second *network* statement places interface S0 on Router A into Area 0. The network statement could have been written more specifically:

RouterA(config)# *router ospf 1*
RouterA(config-router)# *network 172.16.1.2 0.0.0.0 area 1*
RouterA(config-router)# *network 172.17.1.1 0.0.0.0 area 0*

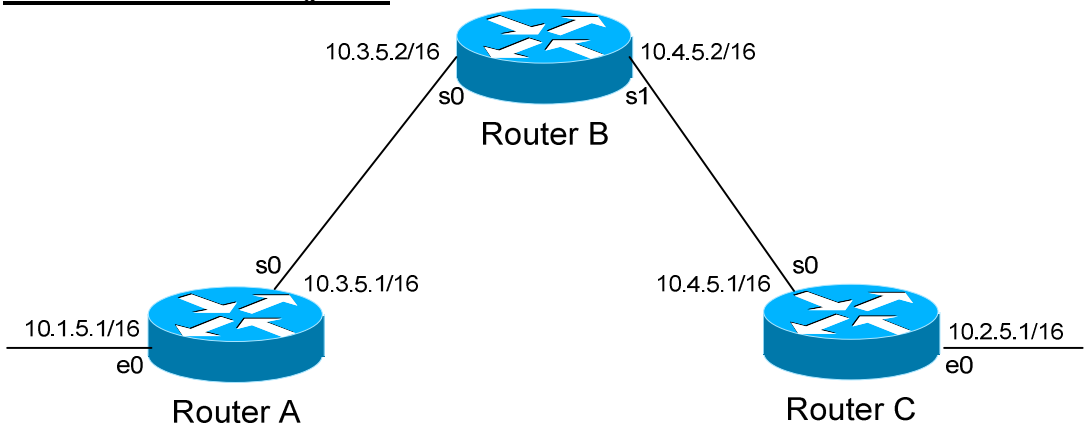
In order for Router B to form a neighbor relationship with Router A, its connecting interface must be put in the same Area as Router A:

RouterB(config)# *router ospf 1*
RouterA(config-router)# *router-id 2.2.2.2*
RouterB(config-router)# *network 172.17.1.2 0.0.0.0 area 0*
RouterB(config-router)# *network 172.18.1.1 0.0.0.0 area 2*

If Router B's S0 interface was placed in a different area than Router A's S0 interface, the two routers would never form a neighbor relationship, and never share routing updates.

* * *

OSPF Passive-Interfaces



It is possible to control which router interfaces will participate in the OSPF process. Just as with EIGRP and RIP, we can use the *passive-interface* command.

However, please note that the *passive-interface* command works differently with OSPF than with RIP or IGRP. OSPF will no longer form neighbor relationships out of a "passive" interface, thus this command prevents updates from being *sent* or *received* out of this interface:

RouterC(config)# *router ospf 1*
RouterC(config-router)# *network 10.4.0.0 0.0.255.255 area 0*
RouterC(config-router)# *network 10.2.0.0 0.0.255.255 area 0*
RouterC(config-router)# *passive-interface s0*

Router C will not form a neighbor adjacency with Router B.

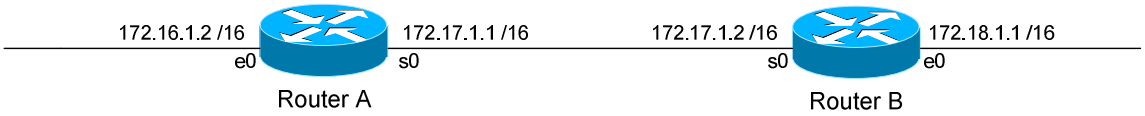
It is possible to configure **all** interfaces to be passive using the *passive-interface default* command, and then individually use the *no passive-interface* command on the interfaces that neighbors **should** be formed on:

RouterC(config)# *router ospf 1*
RouterC(config-router)# *network 10.4.0.0 0.0.255.255 area 0*
RouterC(config-router)# *network 10.2.0.0 0.0.255.255 area 0*
RouterC(config-router)# *passive-interface default*
RouterC(config-router)# *no passive-interface e0*

Always remember, that the *passive-interface* command will prevent OSPF (and EIGRP) from forming neighbor relationships out of that interface. **No** routing updates are passed in either direction.

* * *

OSPF Authentication



OSPF supports authentication to secure routing updates. However, OSPF authentication is configured differently than RIP or EIGRP authentication.

Two forms of OSPF authentication exist, using either **clear-text** or an **MD5 hash.** To configure clear-text authentication, the first step is to enable authentication for the area, under the OSPF routing process:

```
RouterA(config)#  router ospf 1
RouterA(config-router)#  network 172.17.0.0 0.0.255.255 area 0
RouterA(config-router)#  area 0 authentication
```

Then, the authentication *key* must be configured on the interface:

```
RouterA(config)#  interface s0
RouterA(config-if)#  ip ospf authentication
RouterA(config-if)#  ip ospf authentication-key MYKEY
```

To configure MD5-hashed authentication, the first step is also to enable authentication for the area under the OSPF process:

```
RouterA(config)#  router ospf 1
RouterA(config-router)#  network 172.17.0.0 0.0.255.255 area 0
RouterA(config-router)#  area 0 authentication message-digest
```

Notice the additional parameter *message-digest* included with the *area 0 authentication* command. Next, the hashed authentication key must be configured on the interface:

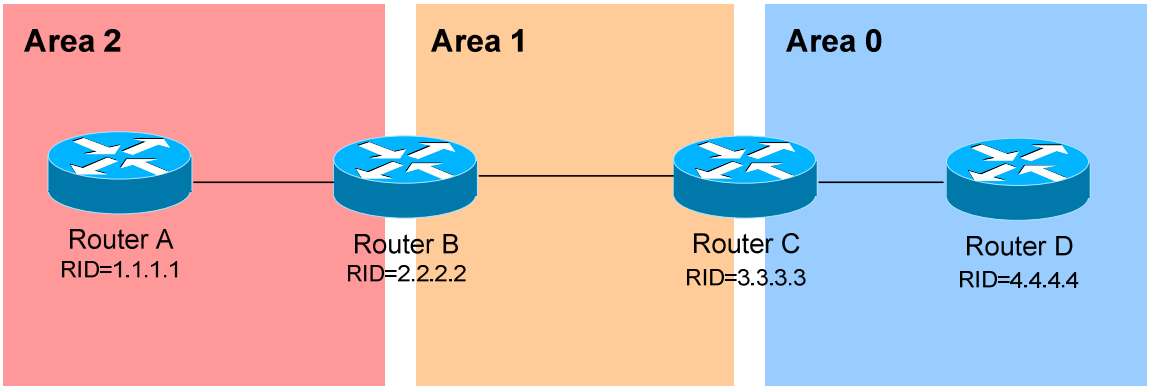
```
RouterA(config)#  interface s0
RouterA(config-router)#  ip ospf message-digest-key 10 md5 MYKEY
```

Area authentication must be enabled on all routers in the area, and the form of authentication must be identical (clear-text or MD5). The authentication keys do *not* need to be the same on every router in the OSPF area, but must be the same on interfaces connecting two neighbors.

Please note: if authentication is enabled for Area 0, the same authentication must be configured on Virtual Links, as they are "extensions" of Area 0.

* * *

OSPF Virtual Links



Earlier in this guide, it was stated that all areas must directly connect into Area 0, as a rule. In the above example, Area 2 has no direct connection to Area 0, but must transit through Area 1 to reach the backbone area. In normal OSPF operation, this shouldn't be possible.

There may be certain circumstances that may prevent an area from directly connecting into Area 0. **Virtual links** can be used as a workaround, to *logically* connect separated areas to Area 0. In the above example, a virtual link would essentially create a tunnel from Area 2 to Area 0, using Area 1 a **transit area**. One end of the Virtual Link *must* be connected to Area 0.

Configuration occurs on the **Area Border Routers (ABRs)** connecting Area 1 to Area 2 (Router B), and Area 1 to Area 0 (Router C). Configuration on Router B would be as follows:

RouterB(config)# *router ospf 1*
RouterB(config-router)# *router-id 2.2.2.2*
RouterB(config-router)# *area 1 virtual-link 3.3.3.3*

The first command enables the *ospf* process. The second command manually sets the *router-id* for Router B to *2.2.2.2*.

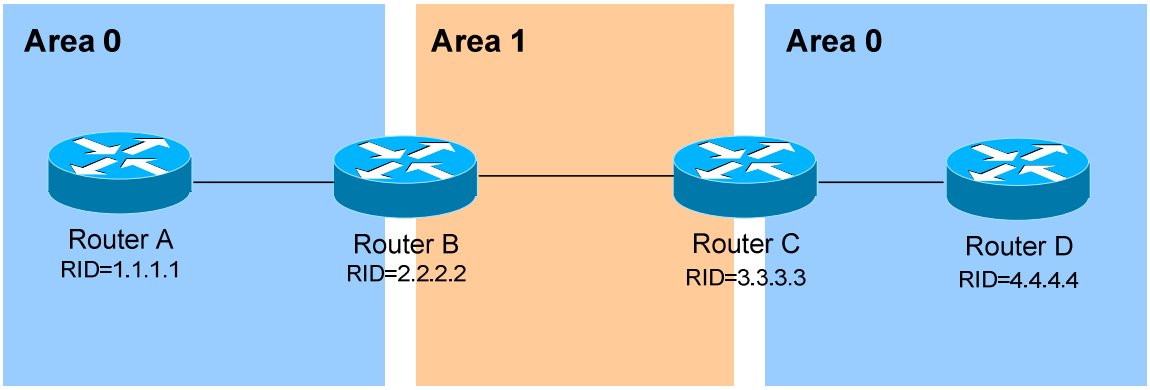
The third command actually creates the *virtual-link*. Notice that it specifies *area 1*, which is the **transit area**. Finally, the command points to the remote ABR's Router ID of *3.3.3.3*.

Configuration on Router C would be as follows:

RouterC(config)# *router ospf 1*
RouterC(config-router)# *router-id 3.3.3.3*
RouterC(config-router)# *area 1 virtual-link 2.2.2.2*

* * *

OSPF Virtual Links (continued)



It is also possible to have two separated (or discontiguous) Area 0's. In order for OSPF to function properly, the two Area 0's must be connected using a **virtual link**.

Again, configuration occurs on the transit area's ABRs:

RouterB(config)# *router ospf 1*
RouterB(config-router)# *router-id 2.2.2.2*
RouterB(config-router)# *area 1 virtual-link 3.3.3.3*

RouterC(config)# *router ospf 1*
RouterC(config-router)# *router-id 3.3.3.3*
RouterC(config-router)# *area 1 virtual-link 2.2.2.2*

Always remember: the area specified in the *virtual-link* command is the **transit** area. Additionally, the transit area ***cannot*** be a stub area.

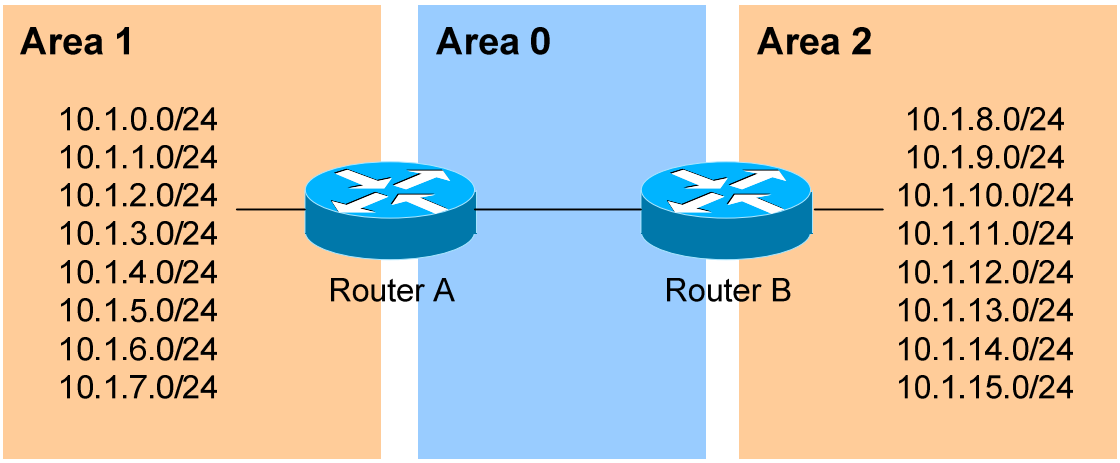
As stated earlier, if authentication is enabled for Area 0, the same authentication must be configured on Virtual Links, as they are "extensions" of Area 0:

RouterB(config)# *router ospf 1*
RouterB(config-router)# *area 1 virtual-link 3.3.3.3 message-digest-key 1 md5 MYKEY*

RouterC(config)# *router ospf 1*
RouterC(config-router)# *area 1 virtual-link 2.2.2.2 message-digest-key 1 md5 MYKEY*

* * *

Inter-Area OSPF Summarization



Consider the above example. OSPF is a classless routing protocol, thus all of the listed networks would be advertised individually. This increases the size of the topology databases and routing tables on routers in the domain, and may be undesirable. Advertising *only* a summary route for inter-area communication can reduce the load on router CPUs.

For example, all of the networks in Area 1 can be summarized as **10.1.0.0/21.** Similarly, all of the networks in Area 2 can be summarized as **10.1.8.0/21.**

Inter-area summarization is configured on **Area Border Routers (ABRs)**. Configuration on Router A would be as follows:

```
RouterA(config)# router ospf 1
RouterA(config-router)# network 10.1.0.0 0.0.7.255 area 1
RouterA(config-router)# area 1 range 10.1.0.0 255.255.248.0
```

The *network* statement includes all of the 10.1.x.0 networks into Area 1. The *area 1 range* command creates a summary route for those networks, which is then advertised into Area 0, as opposed to each route individually.

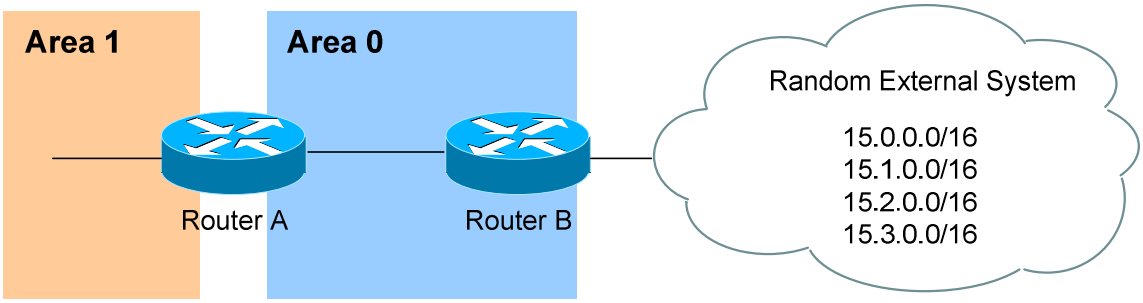
Proper design dictates that a static route be created for the summarized network, pointing to the Null interface. This sends any traffic destined *specifically* to the summarized address to the bit-bucket in the sky, in order to prevent routing loops:

```
RouterA(config)# ip route 10.1.0.0 255.255.248.0 null0
```

In IOS versions 12.1(6) and later, this static route is created automatically.

* * *

External OSPF Summarization



Consider the above example. Router B is an Autonomous System Border Router (ASBR). It is possible to redistribute the four "external" networks into the OSPF system. However, a separate route for each network will be advertised.

Again, this is wasteful. The four external networks can be summarized as **15.0.0.0/14**.

External Summarization is configured on ASBRs, and will only summarize external routes learned by route redistribution. Configuration on Router B would be as follows:

```
RouterB(config)# router ospf 1
RouterB(config-router)# summary-address 15.0.0.0 255.252.0.0
```

This summarized route is now propagated to all routers in every OSPF area.

Summarization can be used to filter certain routes (true route filtering is covered in a separate guide). To force OSPF to advertise the 15.0.0.0 and 15.1.0.0 networks as a summarized route, but *not* advertise the 15.2.0.0 and 15.3.0.0 prefixes:

```
RouterB(config)# router ospf 1
RouterB(config-router)# summary-address 15.0.0.0 255.254.0.0
RouterB(config-router)# summary-address 15.2.0.0 255.255.0.0 not-advertise
RouterB(config-router)# summary-address 15.3.0.0 255.255.0.0 not-advertise
```

The first *summary-address* command summarizes the 15.0.0.0/16 and 15.1.0.0/16 networks to 15.0.0.0/15, and advertises the summary as normal in the OSPF domain. The next two *summary-address* commands specifically reference the 15.2.0.0/16 and 15.3.0.0/16 networks, with the *not-advertise* parameter. As implied, these networks will *not* be advertised in OSPF.

* * *

OSPF Area Types

In order to control the propagation of LSAs in the OSPF domain, several area **types** were developed.

Standard Area – A "normal" OSPF area.

- Routers within a standard area will share Router (Type 1) and Network (Type 2) LSAs to build their topology tables. Once fully synchronized, routers within an area will all have *identical* topology tables.
- Standard areas will accept Network Summary (Type 3) LSAs, which contain the routes to reach networks in all other areas.
- Standard areas will accept ASBR Summary (Type 4) and External (Type 5) LSAs, which contain the route to the ASBR and routes to external networks, respectively.

Configuration of standard areas is straight forward:

Router(config)# *router ospf 1*
Router(config-router)# *network 10.1.0.0 0.0.7.255 area 1*

Stub Area – Prevents external routes from flooding into an area.

- Like Standard areas, Stub area routers will share Type 1 and Type 2 LSAs to build their topology tables.
- Stub areas will also accept Type 3 LSAs to reach other areas.
- Stub areas will ***not accept*** Type 4 or Type 5 LSAs, detailing routes to external networks.

The purpose of Stub areas is to limit the number of LSAs flooded into the area, to conserve bandwidth and router CPUs. The Stub's ABR will *automatically* inject a **default route** into the Stub area, so that those routers can reach the external networks. The ABR will be the *next-hop* for the default route.

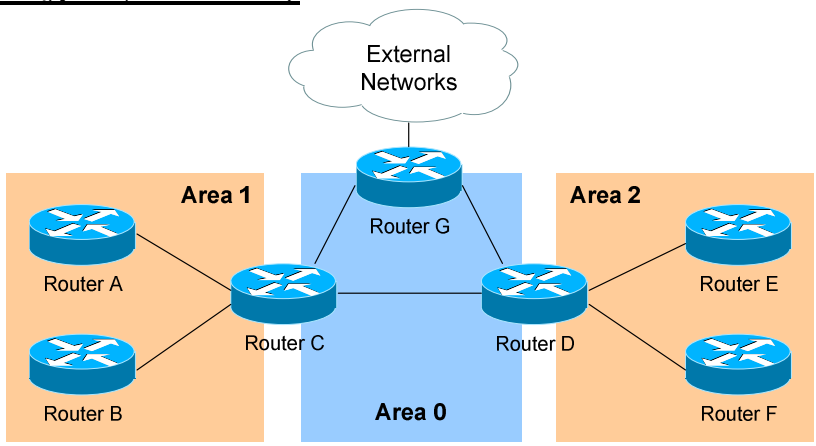
Configuration of stub areas is relatively simple:

Router(config)# *router ospf 1*
Router(config-router)# *network 10.1.0.0 0.0.7.255 area 1*
Router(config-router)# *area 1 stub*

The *area 1 stub* command must be configured on **all** routers in the Stub area. No ASBRs are allowed in a Stub area.

* * *

OSPF Area Types (continued)



Totally Stubby Area – Prevents both inter-area *and* external routes from flooding into an area.

- Like Standard and Stub areas, Totally Stubby area routers will share Type 1 and Type 2 LSAs to build their topology tables.
- Totally Stubby areas will ***not accept*** Type 3 LSAs to other areas.
- Totally Stubby areas will also ***not accept*** Type 4 or Type 5 LSAs, detailing routes to external networks.

Again, the purpose of Totally Stubby areas is to limit the number of LSAs flooded into the area, to conserve bandwidth and router CPUs. The Stub's ABR will instead *automatically* inject a **default route** into the Totally Stubby area, so that those routers can reach both inter-area networks and external networks. The ABR will be the *next-hop* for the default route.

Configuration of totally stubby areas is relatively simple:

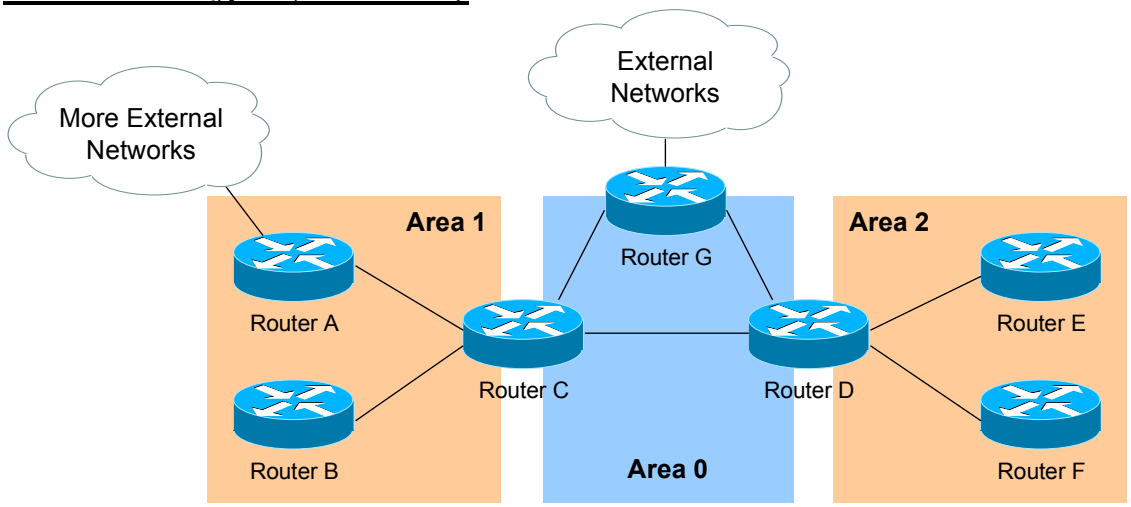
Router(config)# *router ospf 1*
Router(config-router)# *network 10.1.0.0 0.0.7.255 area 1*
Router(config-router)# *area 1 stub no-summary*

The *area 1 stub no-summary* command is configured only on the **ABR** of the Totally Stubby area; other routers within the area are configured with the *area 1 stub* command. No ASBRs are allowed in a Totally Stubby area.

In the above example, if we were to configure Area 1 as a Totally Stubby area, it would not accept any external routes originating from the ASBR (Router G). It *also* would not accept any Type 3 LSAs containing route information about Area 0 and Area 2. Instead, Router C (the ABR) will inject a default route into Area 1, and all routers within Area 1 will use Router C as their gateway to all other networks.

* * *

OSPF Area Types (continued)



***Not So Stubby Area (NSSA)* –** Similar to a Stub area; prevents external routes from flooding into an area, *unless* those external routes originated from an ASBR within the NSSA area.

- Like Standard and Stub areas, NSSA area routers will share Type 1 and Type 2 LSAs to build their topology tables.
- NSSA areas will also accept Network Summary (Type 3) LSAs, which contain the routes to reach networks in all other areas.
- NSSA areas will ***not accept*** Type 4 or Type 5 LSAs, detailing routes to external networks.
- If an ASBR exists *within* the NSSA area, that ASBR will generate **Type 7 LSAs**.

Again, NSSA areas are almost identical to Stub areas. If Area 1 was configured as an NSSA, it would not accept any external routes originating from Router G (an ASBR *outside* Area 1).

However, Area 1 also has an ASBR *within* the area (Router A). Those external routes will be flooded into Area 1 as **Type 7 LSAs**. These external routes *will not* be forwarded to other areas as Type 7 LSAs; instead, they will be converted into Type 5 LSAs by Area 1's ABR (Router C).

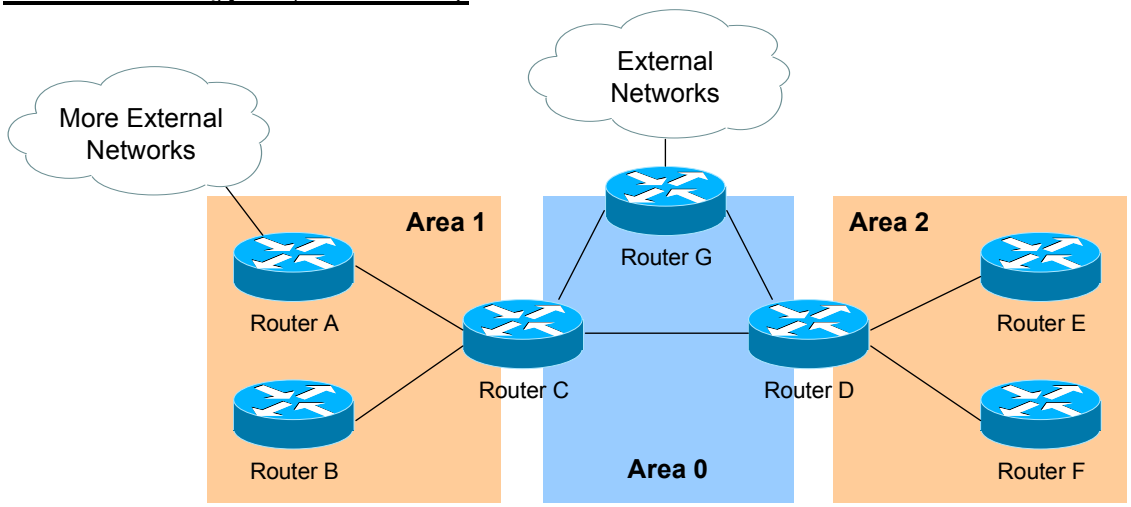
Configuration of NSSA areas is relatively simple:

Router(config)# *router ospf 1*
Router(config-router)# *network 10.1.0.0 0.0.7.255 area 1*
Router(config-router)# *area 1 nssa*

The *area 1 nssa* command must be applied to **all** routers in the NSSA area.

* * *

OSPF Area Types (continued)



Totally Not So Stubby Area (TNSSA) – Similar to a Totally Stubby area; prevents both inter-area *and* external routes from flooding into an area, *unless* those external routes originated from an ASBR within the NSSA area.

- Like Standard and Stub areas, TNSSA area routers will share Type 1 and Type 2 LSAs to build their topology tables.
- TNSSA areas will ***not accept*** Type 3 LSAs to other areas.
- TNSSA areas will ***not accept*** Type 4 or Type 5 LSAs, detailing routes to external networks.
- If an ASBR exists *within* the TNSSA area, that ASBR will generate **Type 7 LSAs**.

With the exception of not accepting inter-area routes, TNSSA areas are identical in function to NSSA areas.

Configuration of TNSSA areas is relatively simple:

Router(config)# *router ospf 1*
Router(config-router)# *network 10.1.0.0 0.0.7.255 area 1*
Router(config-router)# *area 1 nssa no-summary*

The *area 1 nssa no-summary* command is configured only on the **ABR** of the TNSSA area; other routers within the area are configured with the *area 1 nssa* command.

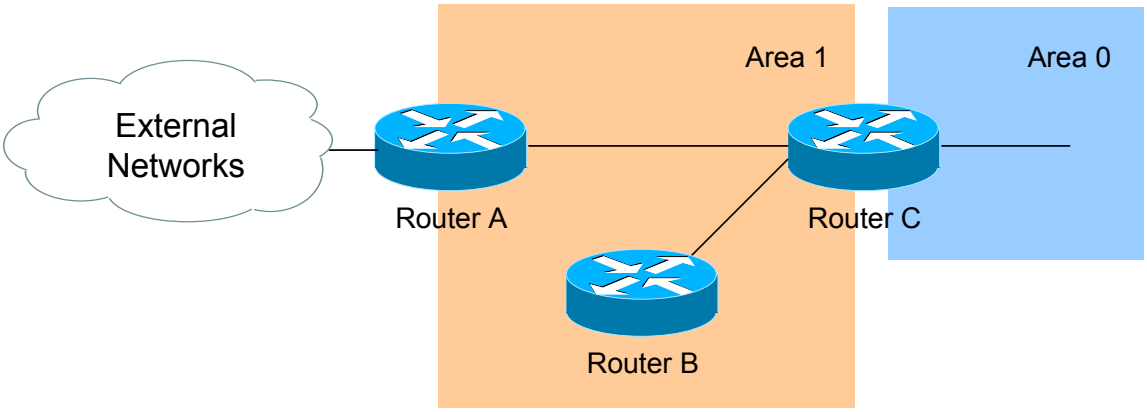
* * *

OSPF and Default Routes

We have learned about four types of OSPF areas:

- *Standard areas*
- *Stub areas*
- *Totally Stubby areas*
- *Not So Stubby areas (NSSA)*

The ABRs and ASBRs of **Standard areas** *do not* automatically generate (or inject) default routes into the area. Consider the following example:



Assume that Area 1 is configured as a Standard area. Router C will forward Type 3 LSAs from all other areas into Area 1, allowing Router A and Router B to reach inter-area networks.

Notice also that Router A is an ASBR, connecting to an external Autonomous System. Thus, Router A will generate Type 5 LSAs, detailing the routes to these external networks.

To additionally force Router A to generate a **default route (**indicating itself as the next hop) for the external networks, and inject this into Area 1. This default route will be advertised as a Type 5 LSA to all other areas:

RouterA(config)# *router ospf 1*
RouterA(config-router)# *default-information originate*

Router A *must* have a default route in its routing table in order for the above command to function. Router A's default route would point to some upstream router in the external Autonomous System.

If a default route *does not* exist in its routing table, Router A can still be forced to advertise a default route using the *always* parameter:

RouterA(config)# *router ospf 1*
RouterA(config-router)# *default-information originate always*

* * *

OSPF and Default Routes (continued)

The ABRs of **Stub** *and* **Totally Stubby** areas *automatically* generate (and inject) a **default route (0.0.0.0/0)** into the area. Routers in Stub areas use this default route to reach *external* networks, while routers in Totally Stubby areas use the default route to reach both *inter-area* and *external* networks.

To control the "cost" metric of the default route in Stub or Totally Stubby areas (configured on the ABR):

Router(config)# *router ospf 1*
Router(config-router)# *area 1 stub*
Router(config-router)# *area 1 default-cost 10*

The ABRs of **NSSA areas** must be *manually configured* to generate (and inject) a default route into the area:

Router(config)# *router ospf 1*
Router(config-router)# *area 1 nssa default-information-originate*

Additionally, the ASBR of an NSSA area can generate and inject a default route. This default route will be advertised as a Type 7 LSA, as Type 5 LSA's are not allowed in NSSAs. The command is no different than injecting a default route from an NSSA ABR:

Router(config)# *router ospf 1*
Router(config-router)# *area 1 nssa default-information-originate*

Reference: (*http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094a74.shtml*)

* * *

OSPF SPF Timers

To adjust the SPF timers in OSPF:

```
Router(config)#  router ospf 1
Router(config-router)#  timers spf 10 15
```

The *timers spf* command includes two parameters, measured in **seconds**. The first (*10*) indicates the SPF-Delay, or how long the OSPF should wait after receiving a topology change to recalculate the shortest path. The second (*15*) indicates the SPF-Holdtime, or how long OSPF should wait *in between* separate SPF calculations.

The *timers spf* command has actually become deprecated. It has been replaced with:

```
Router(config)#  router ospf 1
Router(config-router)#  timers throttle spf 5 10000 80000
```

The *timers throttle spf* command includes three parameters, measure in **milliseconds.** The first *(5)* indicates how long OSPF should wait after receiving a topology change to recalculate the shortest path. The second *(10000)* indicates the hold-down time, or how long OSPF should wait *in between* separate SPF calculations. If OSPF receives another topology change during the hold-time interval, it will continue to *double* the hold-time interval until it reaches the maximum hold-time *(80000*).

The purpose of the both SPF timer commands is to prevent OSPF from constantly converging, if the network links are "flapping." The *timers spf* and *timers throttle spf* commands cannot be used together.

* * *

Advanced OSPF Configuration

To force the OSPF process to ignore OSPF Multicast (Type 6) LSAs:

Router(config)# *router ospf 1*
Router(config-router)# *ignore lsa mospf*

To force an interface to filter all outgoing OSPF LSA's:

Router(config)# *interface e0*
Router(config-if)# *ip ospf database-filter all out*

Loopback interfaces are treated differently than other interfaces, when advertised in OSPF. OSPF will advertise a loopback interface as a specific "host" route (with a mask of /32 or 255.255.255.255). To force OSPF to advertise a loopback interface with its proper subnet mask:

Router(config)# *interface loopback0*
Router(config-if)# *ip address 10.50.5.1 255.255.255.0*
Router(config-if)# *ip ospf network point-to-point*

* * *

Troubleshooting OSPF

To view the OSPF Neighbor Table:

Router# *show ip ospf neighbor*

Neighbor ID	Pri	State	Dead Time	Address	Interface
7.7.7.7	1	FULL/ -	00:00:36	150.50.17.2	Serial0
6.6.6.6	1	FULL/DR	00:00:11	150.50.18.1	Ethernet0

The Neighbor Table provides the following information about each neighbor:

- The ***Router ID*** of the remote neighbor.
- The OSPF ***priority*** of the remote neighbor (used for DR/BDR elections).
- The current neighbor ***state***.
- The ***dead interval*** timer.
- The connecting ***IP address*** of the remote neighbor.
- The local ***interface*** connecting to the remote neighbor.

To view the OSPF topology table:

Router# *show ip ospf database*

OSPF Router with ID (9.9.9.9) (Process ID 10)

Router Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum	Link count
7.7.7.7	7.7.7.7	329	0x80000007	0x42A0	2
8.8.8.8	8.8.8.8	291	0x80000007	0x9FFC	1

Summary Net Link States (Area 0)

Link ID	ADV Router	Age	Seq#	Checksum
192.168.12.0	7.7.7.7	103	0x80000005	0x13E4
192.168.34.0	7.7.7.7	105	0x80000003	0x345A

The Topology Table provides the following information:

- The actual ***link*** (or *route*).
- The ***advertising*** Router ID.
- The link-state ***age*** timer.
- The ***sequence number*** and ***checksum*** for each entry.

(Reference: http://www.cisco.com/en/US/products/sw/iosswrel/ps5187/products_command_reference_chapter09186a008017d02e.html)

* * *

Troubleshooting OSPF (continued)

To view the specific information about an OSPF process:

Router# *show ip ospf 1*

```
Routing Process "ospf 1" with ID 9.9.9.9
 Supports only single TOS(TOS0) routes
 Supports opaque LSA
 SPF schedule delay 5 secs, Hold time between two SPFs 10 secs
 Minimum LSA interval 5 secs. Minimum LSA arrival 1 secs
 Number of external LSA 0. Checksum Sum 0x0
 Number of opaque AS LSA 0. Checksum Sum 0x0
 Number of DCbitless external and opaque AS LSA 0
 Number of DoNotAge external and opaque AS LSA 0
 Number of areas in this router is 1. 1 normal 0 stub 0 nssa
 External flood list length 0
   Area BACKBONE(0)
      Number of interfaces in this area is 1
      Area has no authentication
      SPF algorithm executed 3 times
      Area ranges are
      Number of LSA 2. Checksum Sum 0xDDEC
      Number of opaque link LSA 0. Checksum Sum 0x0
      Number of DCbitless LSA 0
      Number of indication LSA 0
      Number of DoNotAge LSA 0
      Flood list length 0
```

The *show ip ospf* command provides the following information:

- The local ***Router ID.***
- ***SPF Scheduling*** information, and various ***SPF timers.***
- The number of ***interfaces*** in specific ***areas***, including the ***type*** of area.
- The link-state ***age*** timer.
- The ***sequence number*** and ***checksum*** for each entry.

* * *

Troubleshooting OSPF (continued)

To view OSPF-specific information on an interface:

Router# *show ip ospf interface s0*

```
Serial0 is up, line protocol is up
  Internet Address 192.168.79.2/24, Area 0
  Process ID 10, Router ID 9.9.9.9, Network Type POINT_TO_POINT, Cost: 64
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:04
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 7.7.7.7
  Suppress hello for 0 neighbor(s)
```

The *show ip ospf interface* command provides the following information:

- The local ***Router ID***.
- The interface ***network type***.
- The OSPF ***cost*** for the interface.
- The interface ***Hello*** and ***Dead*** timers.
- A list of neighbor ***adjacencies***.

To view routing protocol specific information for OSPF:

Router# *show ip protocols*

```
Routing Protocol is “ospf 10"
  Invalid after 0 seconds, hold down 0, flushed after 0
  Outgoing update filter list for all interfaces is
  Incoming update filter list for all interfaces is
  Routing for Networks:
    192.168.79.0  0.0.0.255 area 0
    192.168.109.0 0.0.0.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
    7.7.7.7          110            00:01:05
  Distance: (default is 110)
```

The *show ip protocols* command provides the following information:

- Locally originated ***networks*** that are being advertised.
- Neighboring ***sources*** for routing information
- The ***administrative distance*** of neighboring sources.

* * *

Troubleshooting OSPF (continued)

To reset an OSPF process, including neighbor adjacencies:

Router# *clear ip ospf process*

To display information about OSPF virtual-links:

Router# *show ip ospf virtual-links*

To display routes to both ABRs and ASBRs:

Router# *show ip ospf border-routers*

To debug OSPF in realtime:

Router# *debug ip ospf adj*
Router# *debug ip ospf events*
Router# *debug ip ospf hello*

* * *

- 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.

* * *

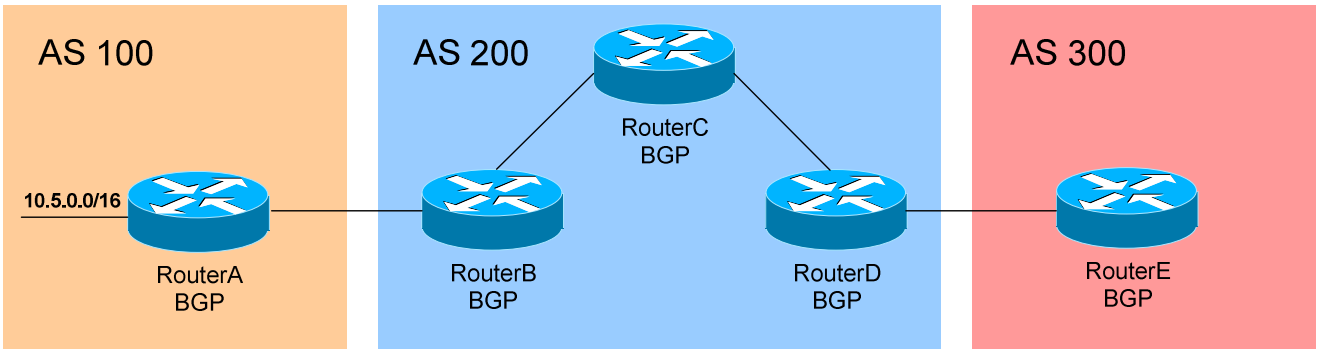
BGP Peers (Neighbors)

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.

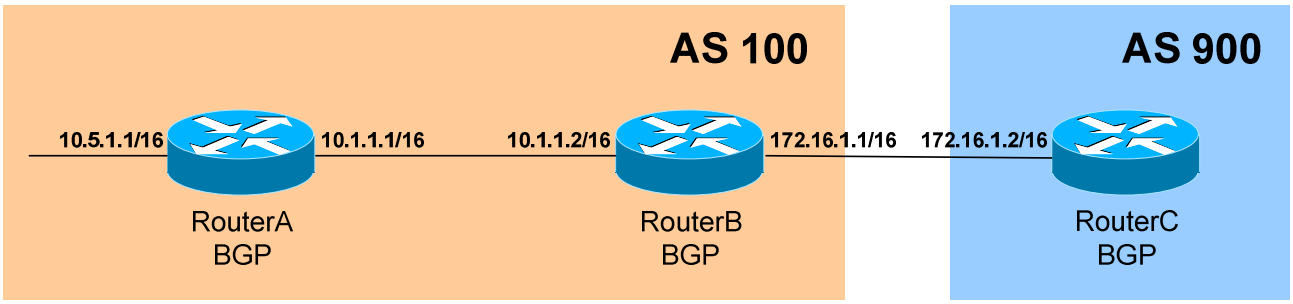
As a BGP peer session is forming, it will pass through several **states**. This process is known as the BGP **Finite-State Machine (FSM)**:

- **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*

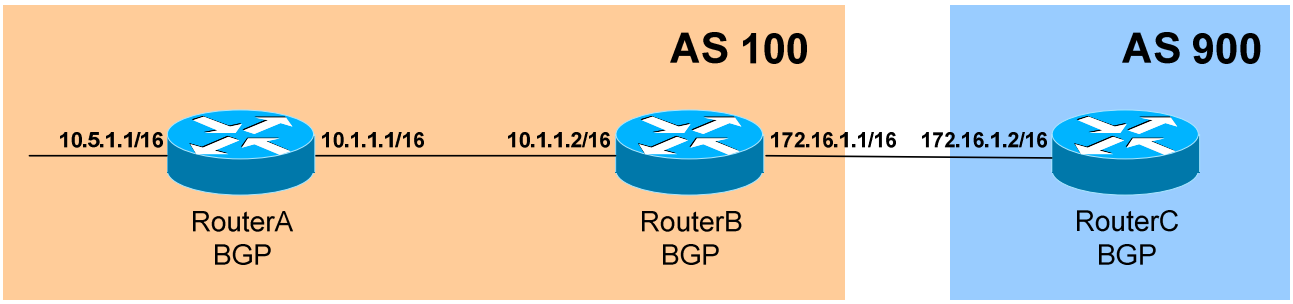
RouterC must then point to RouterB's loopback (assume the address is 1.1.1.1/24) in its neighbor statement:

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*

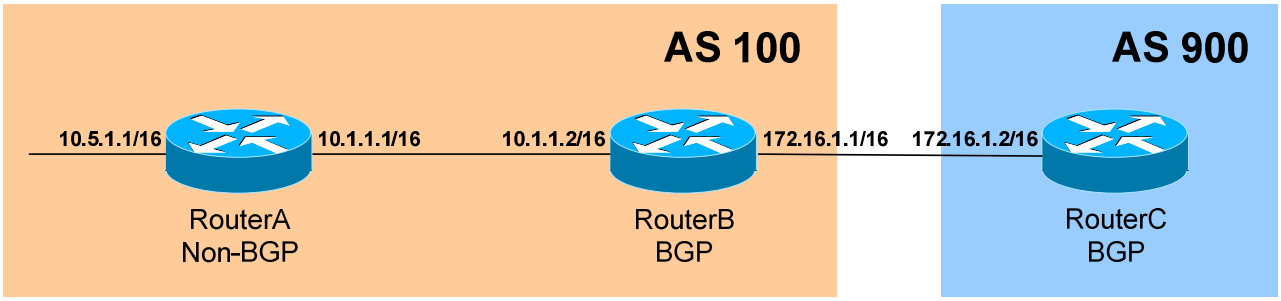
The above command sets the Keepalive timer to *30* seconds, and the Hold-time timer to *90* seconds. If the configured Hold-time timers between two peers are different, the peer session **will still** be established, and the **smallest** timer value will be used.

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:

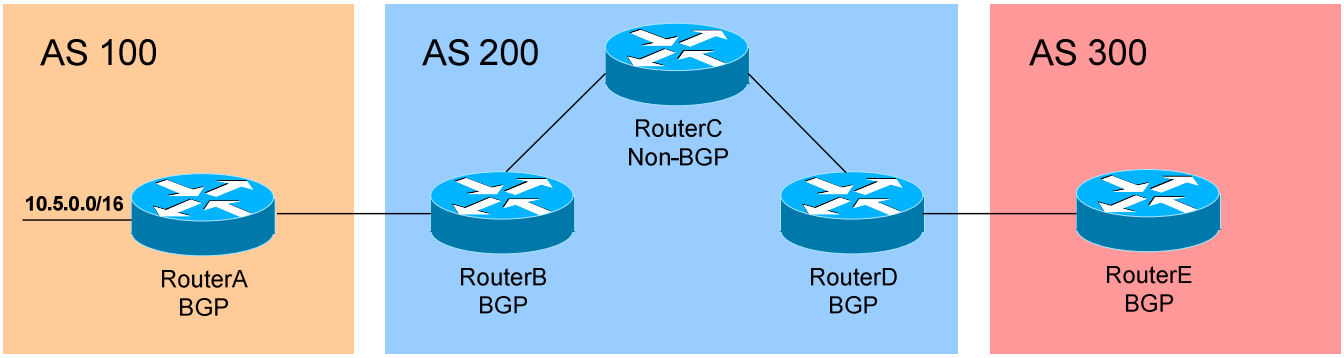
RouterB# *show ip bgp neighbors*

```
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.

* * *

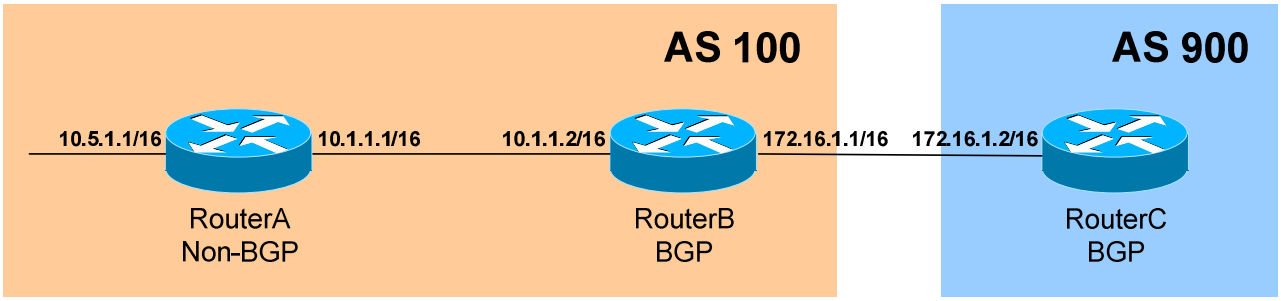
Originating Prefixes in BGP

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:

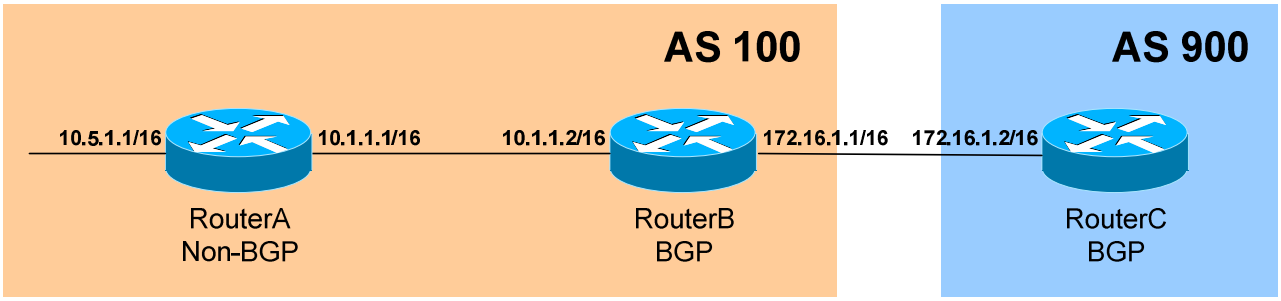
```
RouterB(config-router)# network 10.5.0.0 mask 255.255.255.0
```

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*

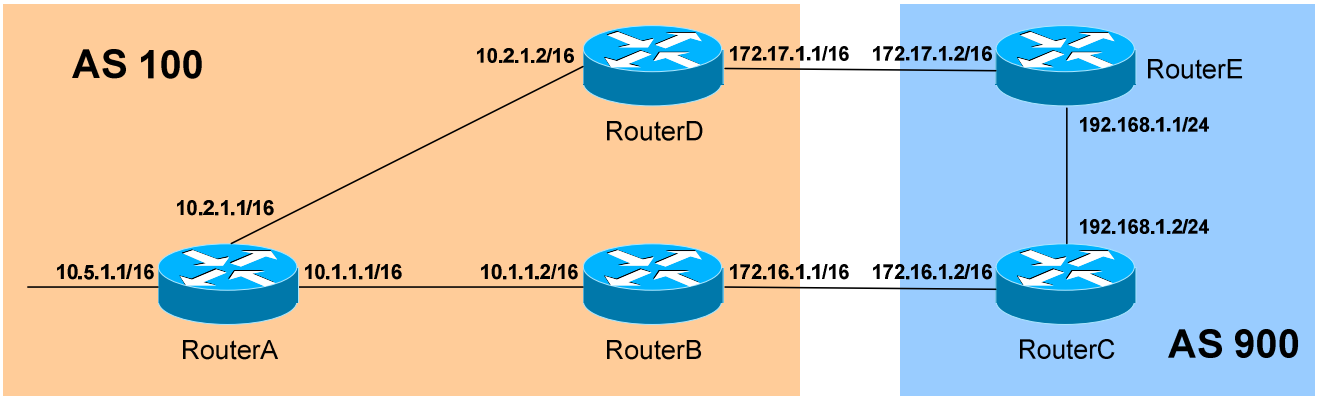
```
   Network          Next Hop          Metric LocPrf Weight Path
*> 10.5.0.0         172.16.1.1             0     100      0    100 i
```

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.

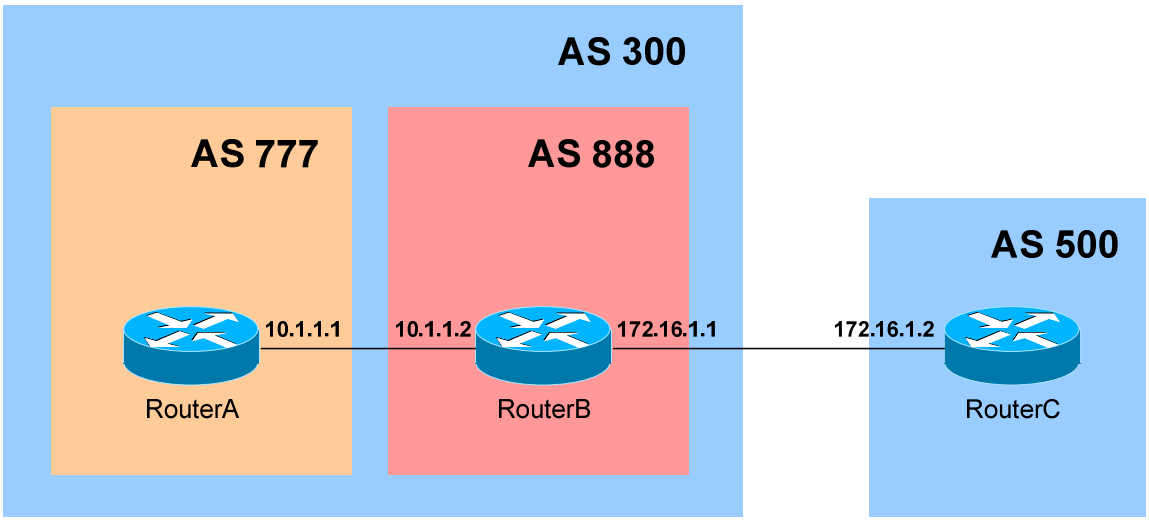
All route-reflector specific configuration takes place on the route reflector server:

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?

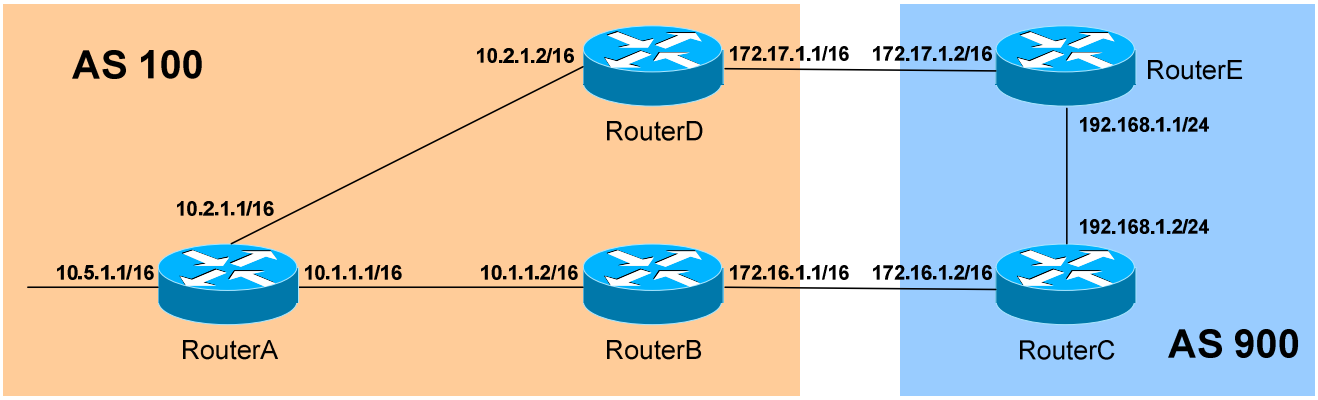
When applying attributes, Weight and Local Preference are applied to *inbound* routes, dictating the best *outbound* path.

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*

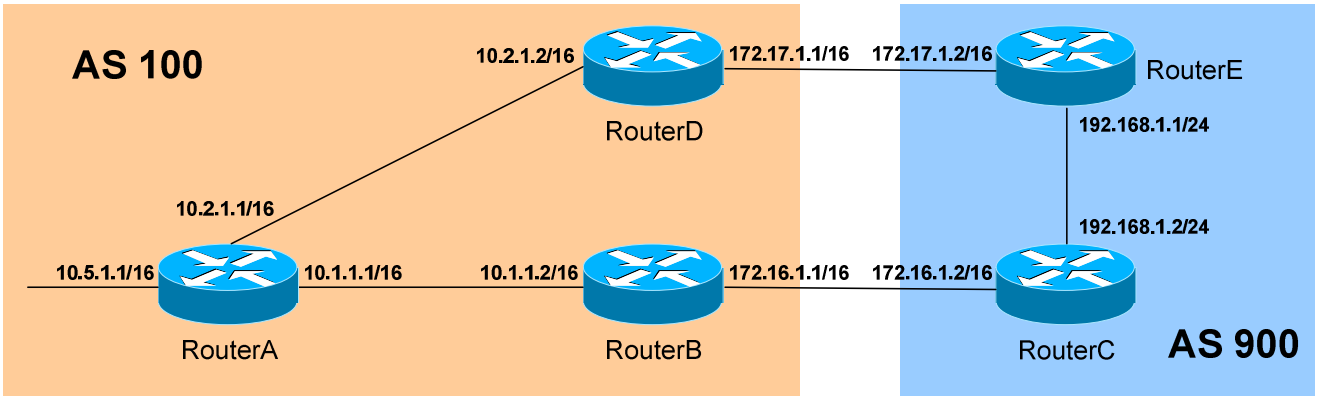
Finally, the route-map is applied to the preferred neighbor:

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*

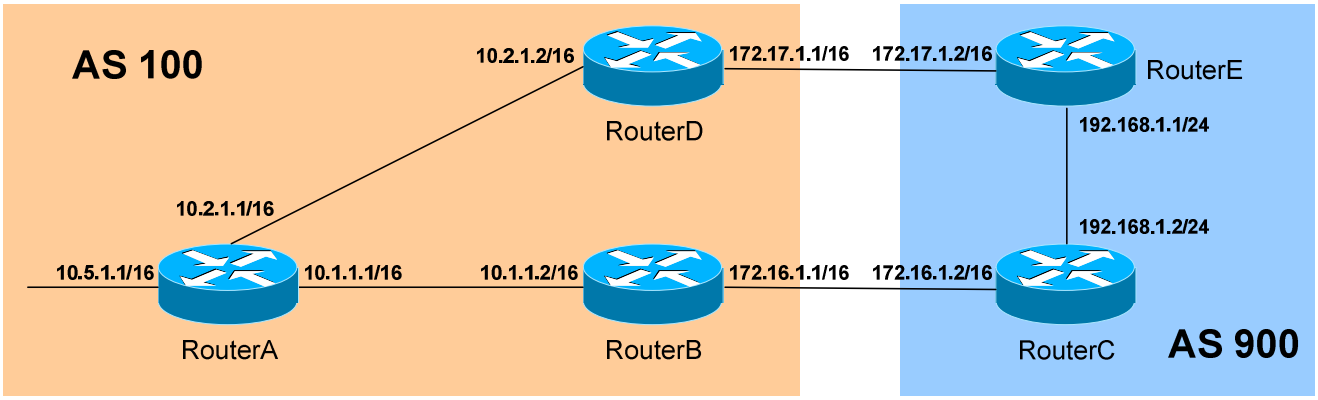
RouterD(config)# *route-map PREFERENCE permit 10*
RouterD(config-route-map)# *match ip address prefix-list MYLIST*
RouterD(config-route-map)# *set local-preference 300*

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:

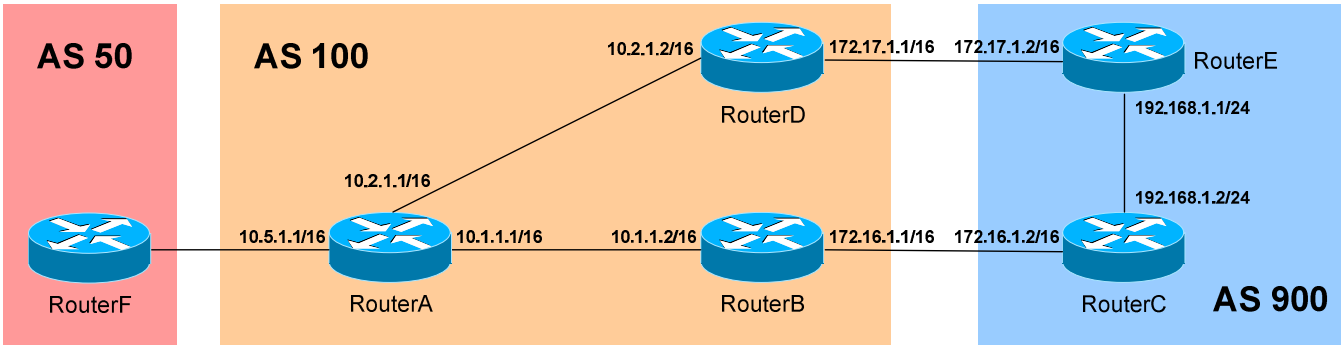
RouterC# *show ip bgp*

```
   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*

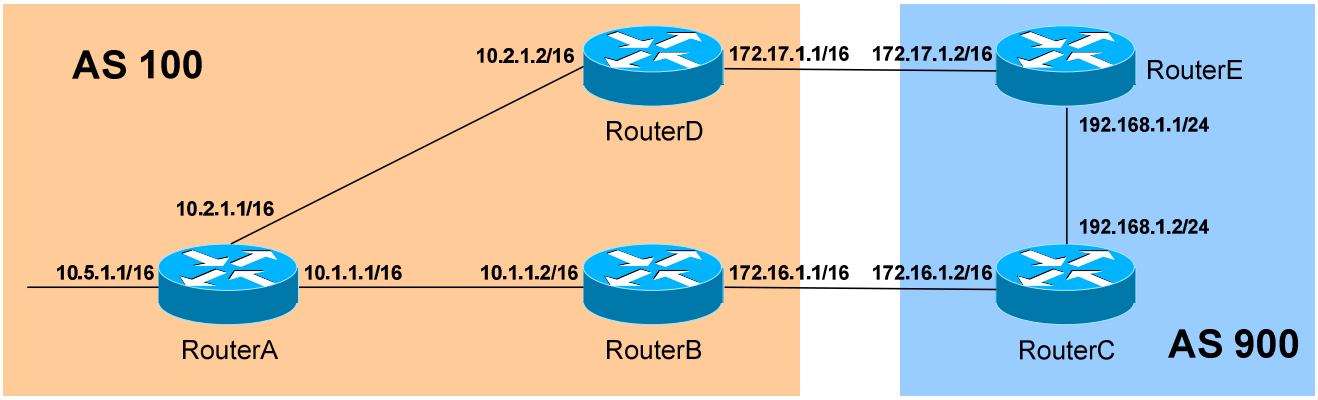
To view what BGP routing entries the AS-Path access-list will match:

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.

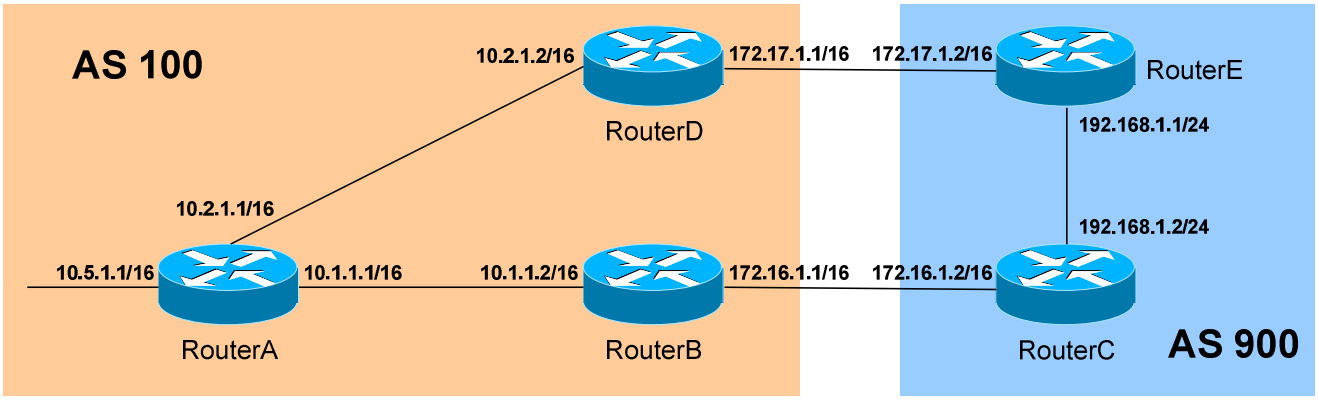
When viewing the BGP routing table, the origin code is listed at the *end* of each line in the table:

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 200*

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*

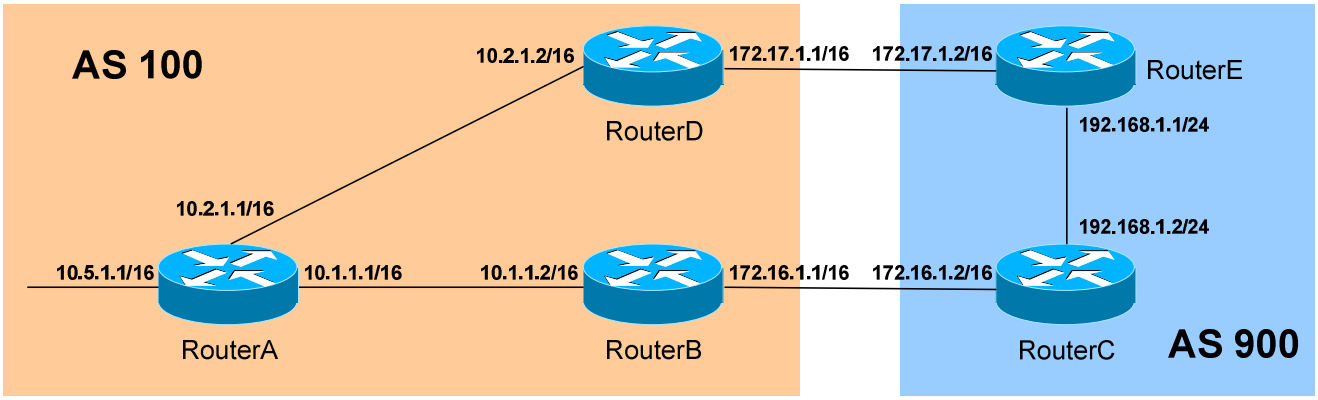
```
   Network          Next Hop        Metric LocPrf Weight Path
*  10.5.0.0         172.16.1.1         200    100      0   100 i
*> 10.5.0.0         172.17.1.1           0    100      0   100 i
```

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.

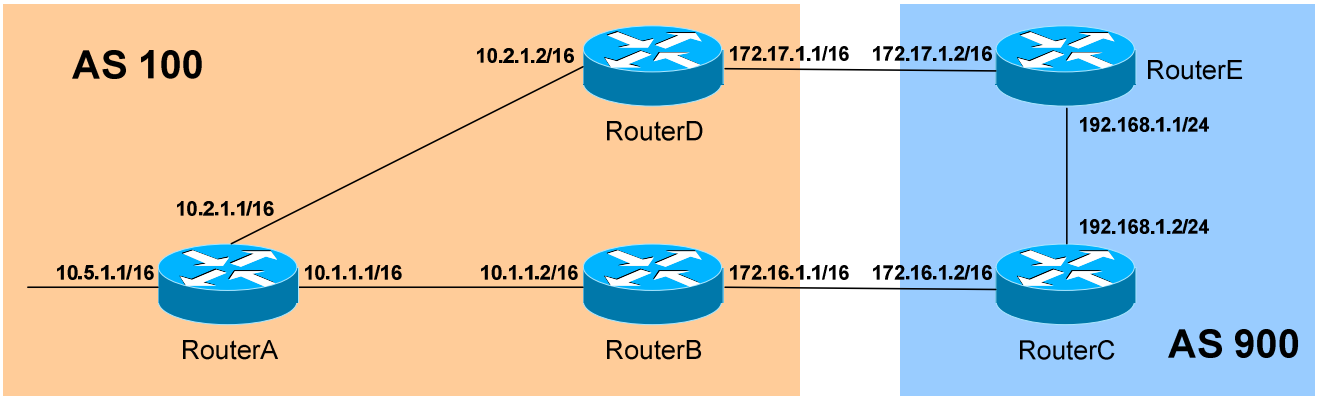
RouterE(config)# *router bgp 100*
RouterE(config-router)# *bgp always-compare-med*

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*

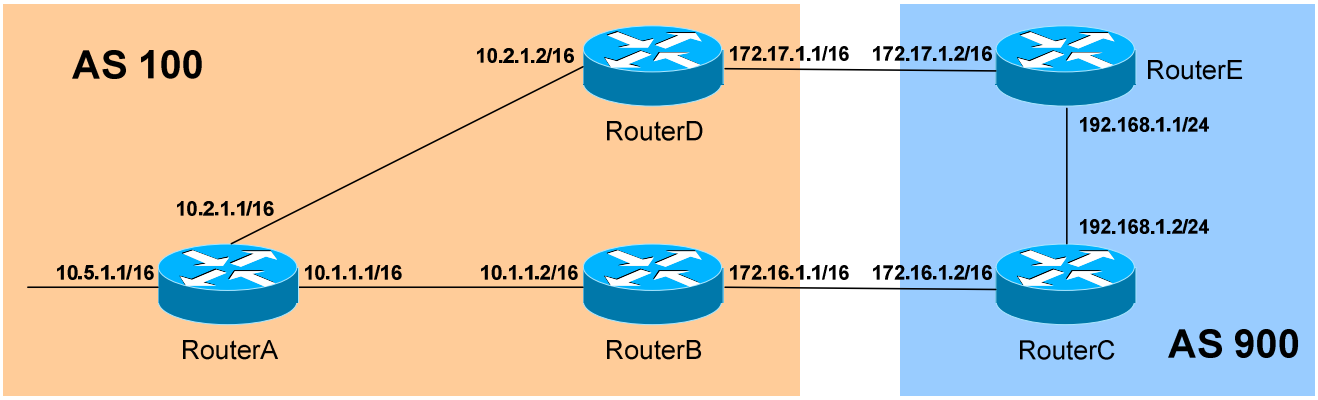
Additionally, there are four **well-known** communities that can be referenced by name:

- **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.

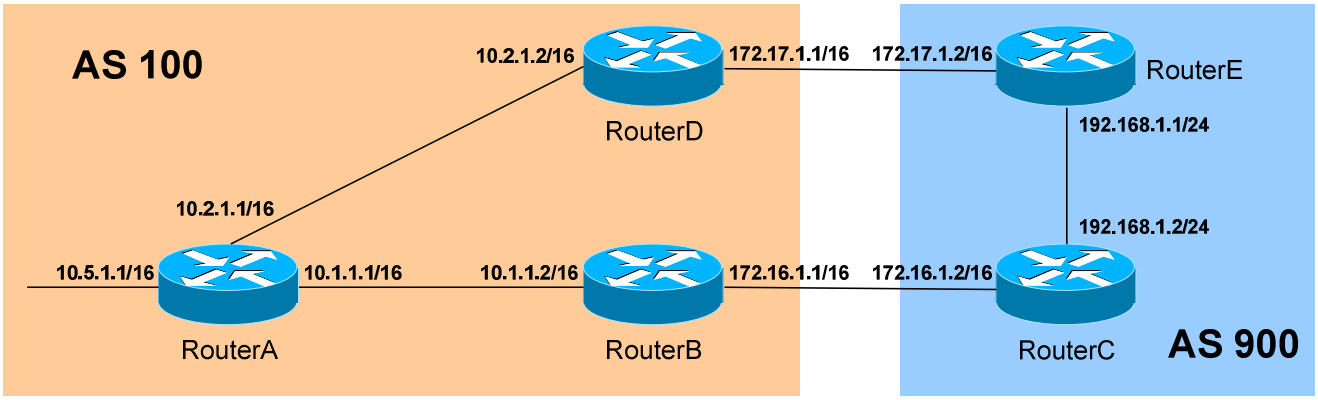
Finally, route-dampening must be enabled under the BGP process:

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*