## Riverstone Networks BGP Support Page

Exterior Gateway Protocol (EGP)

Interior Gateway Protocol (IGP)

IP Prefix

Network Layer Reachability Information (NLRI)

Path Attribute

Path Vector Algorithm

Route Flap

# *BGP Fundamentals*

A Border Gateway Protocol (BGP) primer that provides an in-depth overview of each component of the BGP protocol. BGP message types, the finite state machine and path attributes are discussed to provide the necessary knowledge of how the protocol works.

## Introduction and Background

BGP first became an Internet standard in 1989 and was originally defined in RFC 1105. It was then adopted as the EGP of choice for inter-domain routing. The current version, BGP-4, was adopted in 1995 and is defined in RFC 1771. BGP-4 supports Classless Inter Domain Routing (CIDR) and is the routing protocol that people use in today to route between autonomous systems. It has proven to be scalable, stable and provides the mechanisms needed to support complex routing policies. When people talk about "BGP" today, they implicitly mean BGP-4. There is no need to specify the –4 version number because no one uses earlier versions, and very few vendors even still support them.

BGP continues to evolve through the Internet standards process work at the IETF, the latest draft version is draft-ietf-idr-bgp4-17.txt. As the Internet routing requirements change, BGP is extended to continue to provide these knobs needed to control routing information and to support the new requirements. The base RFC has been extended by several RFCs and I-Ds that define new features.

Riverstone's RapidOS also continues to evolve, as we implement support for the latest extensions that are being defined. For a full list of supported features by version, please consult the Appendix.

## Protocol Mechanics

BGP uses TCP to establish a reliable connection between two BGP speakers on port 179. Exactly one TCP session is established between each peer for each BGP session. No routing information can be exchanged until the TCP session has been established. This implies that each BGP speaker must have working IP connectivity between them first, which is usually provided by a directly connected interface or the IGP. For added security, MD5 signatures can be used to authenticate each TCP segment.

We call BGP a path vector protocol, because it stores routing information as a combination of a destination and attributes of the path to that destination. The protocol uses a deterministic route selection process to select the best route from multiple feasible routes, using the path attributes as criteria. Characteristics such as delay, link utilization or router hops are not considered in this process. The route selection process is key to understanding and implementing BGP policies and is discussed in its own section on the RS Routing Model.

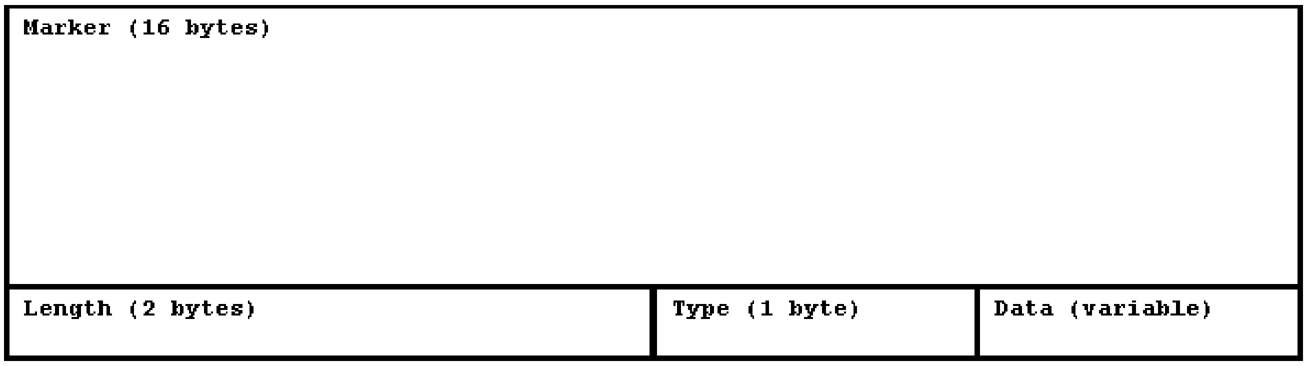
Unlike most IGP protocols, BGP only sends a full routing update once when a BGP session is established and then only sends incremental changes.  BGP only recalculates routing information relative to these updates, there is no regular process that must update all of its routing information like the SPF calculations in OSPF or IS-IS.  Although IGP convergence may be faster, an IGP will simply not scale to support the number of routes needed for inter-domain routing.  IGPs also lack the path attributes that BGP carries, which are essential for selecting the best route and building routing policies.  BGP is the only protocol that is suitable for use between autonomous systems, because of the inherent support for routing policies that the path attributes provide.  These policies allow you to accept, reject or change routing information before it is used to make forwarding decisions.  This ability gives network operators a high degree of protection from accepting routing information they might not want, and the ability to control routing information according to their particular needs.  Neither OSPF or IS-IS provide policies to reject or change routing information, and thus should not be run between autonomous systems.

BGP runs in two modes: EBGP and IBGP.  EBGP (Exterior BGP) is run between different autonomous systems, and IBGP (Interior BGP) is run between BGP routers in the same autonomous system.  The behavior of the BGP speaker is different in each of these modes and these differences are further discussed in the Path Attributes section.

## Message Types

Five message types are used by BGP to negotiate parameters, exchange routing information and indicate errors.  Each message can be between 19 and 4096 bytes long, and relies on TCP/IP for delivery, sequencing and fragmentation.  This implies that multiple BGP messages can be sent in one TCP segment.  A common 19-byte message header is used for each message, and certain messages also contain data depending on the message type.  The Type-Length-Value (TLV) format is often used to provide flexibility, extensibility and ease in processing of the messages and their data.

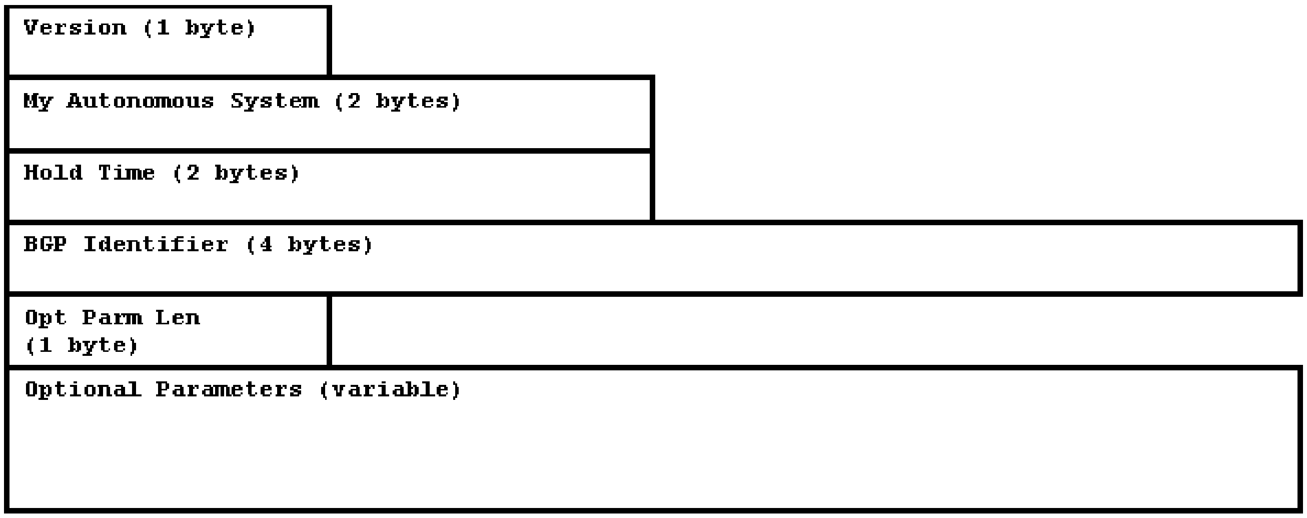
### BGP Message Header



The BGP message header is used in all messages, and contains the following fields.

- *Marker* (16 bytes): contains all 1s (0xFF) and is used for synchronization when multiple messages are in a TCP segment
- *Length* (2 bytes): total message length
- *Type* (1 bytes): message type
- *Data* (variable): depending on the message type, data may or may not be present

### OPEN Message (Type 1 – RFC 1771)

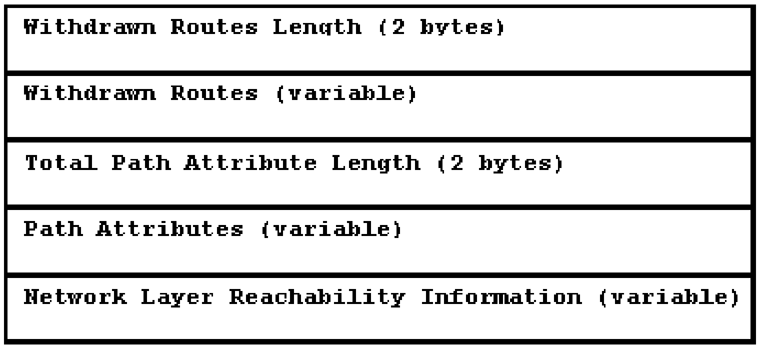


The first BGP message that is sent after the TCP connection has been established is the OPEN message.  It is used to exchange configuration information and to negotiate common parameters for the peering session.  It contains the following fields.

- *Version* (1 byte): BGP version, defaults to BGP-4 and cannot be changed
- *My Autonomous System* (2 bytes): AS number of the BGP speaker
- *Hold Time* (2 bytes): number of seconds that can elapse without receipt of an UPDATE or KEEPALIVE message before the peer is assumed to be down (Riverstone’s default is 180 seconds)

- *BGP Identifier* (4 bytes): sender’s BGP ID, equal to the router ID
- *Optional Parameter Length* (1 byte): length is set to 0 if none are present
- *Optional Parameters* (variable):
  - Header password authentication (RFC 1771), defined in the RFC but not implemented by any vendor
  - Capabilities advertisement (RFC 2842), provides a mechanism to negotiate the capability to use various BGP features

## UPDATE Message (Type 2 – RFC 1771)


Withdrawn Routes Length (2 bytes)
Withdrawn Routes (variable)
Total Path Attribute Length (2 bytes)
Path Attributes (variable)
Network Layer Reachability Information (variable)

UPDATE messages are used to distribute the routing information in BGP, and are only sent after the session is established.  An UPDATE message can be used to withdraw existing routes, advertise new routes, or both.

In this message, 2-tuples consisting of a mask length and prefix are used to represent IP prefixes.  These prefixes are used to communicate which routes are to be withdrawn, or which routes are to be advertised.  When routes are advertised, the prefixes are called Network Layer Reachability Information (NLRI).  Often, several path attributes are shared among the same prefixes, for example if they all are originated by the same BGP speaker. In this case, BGP will send the path attributes and the associated prefixes as NLRI in the same message. This makes sending and storing the routing information much more efficient. Similarly, multiple IP prefixes can be withdrawn at the same time.

The following fields comprise the UPDATE message.

- *Withdrawn Routes Length* (2 bytes): length of withdrawn routes field, a length of 0 means there are no withdrawn routes
- *Withdrawn Routes* (variable): 2-tuple IP prefix(es) of unfeasible route(s) to be withdrawn
- *Total Path Attribute Length* (2 bytes): total length of the path attribute information, a length of 0 means there is no NLRI
- *Path Attributes* (variable): path attributes represented as variable length TLVs
- *NLRI* (variable): 2-tuple IP prefix(es) of reachability information

**KEEPALIVE Message (Type 3 – RFC 1771)**

KEEPALIVE messages are sent periodically at 1/3 the *Hold Time* to indicate that a peer is still operating normally to keep the BGP session alive (Riverstone's default is 60 seconds). This message only contains the BGP header and no data.

**NOTIFICATION Message (Type 4 – RFC 1771)**

| | | |
|---|---|---|
| Error Code (1 byte) | Error Subcode (1 byte) | Data (variable) |

The NOTIFICATION message is sent when BGP detects an error condition, after which the peering session is terminated and the TCP is connection is closed. The cause of the error condition is sent to the peer for debugging and troubleshooting. Error codes are defined in RFC 1771 and indicate exactly what the problem is through an error code and subcode. A NOTIFICATION message contains the following fields.

- *Error Code* (1 byte): type of error
- *Error Subcode* (1 byte): indicates more details about the error
- *Data* (variable): optional error data

**ROUTE-REFRESH Message (Type 5 – RFC 2918)**

| | | |
|---|---|---|
| AFI (2 bytes) | Reserved (1 byte) | SAFI (1 byte) |

The ROUTE-REFRESH message is not defined in RFC 1771, but as a BGP capability in RFC 2918. However, it has been so widely implemented that is only makes sense to discuss this message in this section with the other BGP messages. Route refresh is used to request a complete retransmission of a peer's routing information without tearing down and reestablishing the BGP session. Using this feature, routing policy changes can be made without storing an unmodified copy of the peer's routes on the local router, which in turn saves RAM and resources. The ROUTE-REFRESH message was designed to be protocol independent. Thus you can request a retransmission of a peer's IPv4 unicast routes only, but none of it's IPv6 routes, for example. ROUTE-REFRESH messages use the following fields.

- *AFI* (2 bytes): Address Family Identifier, for example IPv4 or IPv6
- *Reserved* (1 byte): always set to 0
- *SAFI* (1 byte): Subsequent Address Family Identifier, for example unicast or multicast

---

# BGP Fundamentals

## Finite State Machine

A finite state machine is an abstract model of a machine that has a finite number of defined states in which it can exist at any time.  Events such as timers or external inputs cause transitions between states.  A light switch is a simple FSM that has two states: on and off.  The external input, flipping the switch, causes a state transition between on and off.  BGP uses a simple FSM to figure out what to do next when something happens, such as receiving a message or a timer firing that reminds it to take a certain action.  One FSM is maintained for each BGP session, which allows sessions to operate independently of each other.

## Finite State Machine Timers

BGP uses five timers that are used to cause state transitions, and each values is stored in units of seconds.  Most are not configurable in BGP implementation, but you should be aware that they exist.  In RapidOS, only the *Hold Time* is configurable.

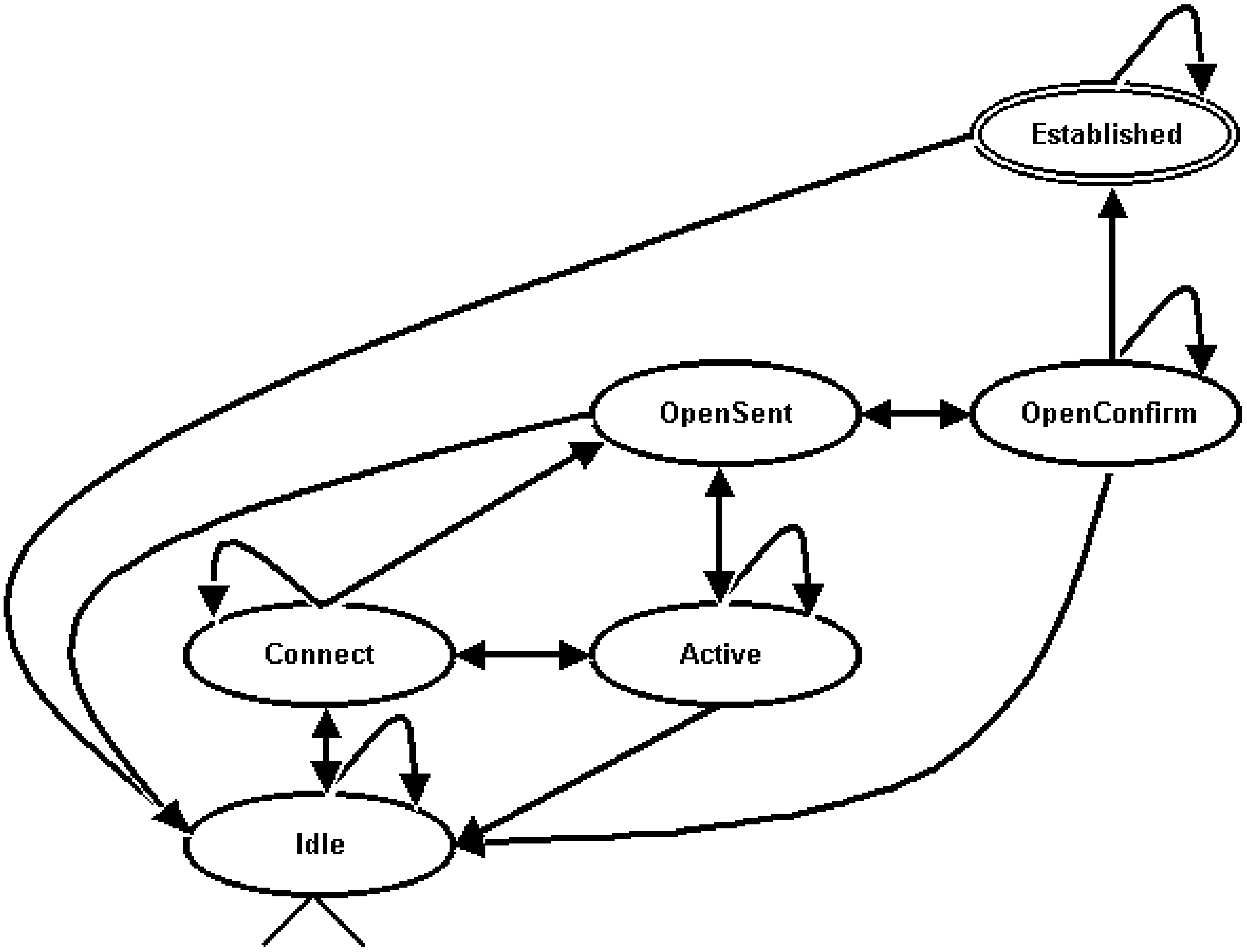
- *ConnectRetry*: used only when BGP is trying to establish a TCP connection to its peer, and determines how often a TCP connection is initiated
- *Hold*: number of seconds that can elapse without receipt of an UPDATE or

KEEPALIVE message before the peer is assumed to be down (Riverstone's default is 180 seconds)

- *KeepAlive*: used to generate KEEPALIVE messages at a rate of 1/3 the *Hold Time* (Riverstone's default is 60 seconds)
- *MinASOriginationInterval*: used to throttle how often internal changes within the AS are sent
- *MinRouteAdvertisementInterval*: used to throttle how often changes to the same route are sent

The first three timers are used and described extensively in the next section, but the last two should be explained with more detail as they are internal to each BGP implementation and not generally discussed.  These timers help reduce the propagation of rapid changes that can cause routing instability.  It may take a few seconds before a route is advertised, but stability is gained by waiting just a little bit.

## Finite State Machine States



The BGP FSM is depicted in the above picture.  Each circle represents a BGP state, and the arcs represent state transitions.  The arrow direction indicates state transition events between states.  Note that it is possible to only make a transition in a single direction between certain states.  The *Idle* state is the initial state, and the *Established* state is the accepting, or end state.  This diagram does not show all of the state transition events, in order to simplify it.  Each state will now be explained in detail, to bring the timers, state diagram and each state together into a complete description of how BGP operates.

## *Idle* State

In the initial state, BGP is waiting for a *Start* event to jumpstart the BGP session. *Start* events are generated automatically by the router when BGP is *Idle*. When the Start event is received, BGP transitions to *Connect*. A manual clear or error in any state will return BGP back to *Idle* where it waits for the *Start* event again. The state of each peer is shown in the CLI output of "`bgp show summary`".

```
RS# bgp show summary
Local router ID is 192.168.0.1, Local AS number 65030
BGP Route Entries 0, Unique AS Paths 2
Unique Communities 0, Unique Extended Communities 0

Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes
Rcvd/Sent
--------        -    -- ------- -------      ------- ---------
--------
[Group Id: VVNet]
10.0.0.2        0 65015       0       0
                Idle
```

## *Connect* State

BGP is initiating a TCP connection to its peer when in *Connect*. If the TCP session is established, a BGP OPEN message is sent and BGP transitions to *OpenSent*. If the *ConnectRetry* timer expires before the TCP connection is accepted, BGP stays in *Connect* and initiates a TCP connection to its peer. If the TCP connection cannot be made, BGP transition to *Active*.

```
RS# bgp show summary
Local router ID is 192.168.0.1, Local AS number 65030
BGP Route Entries 0, Unique AS Paths 2
Unique Communities 0, Unique Extended Communities 0

Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes
Rcvd/Sent
--------        -    -- ------- -------      ------- ---------
```

```
    --------
    [Group Id: VVNet]
    10.0.0.2        0 65015       0       0
    Connect
```

### *Active* State

BGP is listening for a TCP connection from its peer. If the TCP session is established, a BGP OPEN message is sent and BGP transitions to *OpenSent*. If the *ConnectRetry* timer expires, BGP initiates a TCP connection to its peer and transitions to *Connect*. This does not mean that BGP is "up", but rather that BGP is active and willing to accept a connection from its peer.

```
    RS# bgp show summary
    Local router ID is 192.168.0.1, Local AS number 65030
    BGP Route Entries 0, Unique AS Paths 2
    Unique Communities 0, Unique Extended Communities 0

    Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes
    Rcvd/Sent
    --------        -    -- ------- -------      ------- ----------
    --------
    [Group Id: VVNet]
    10.0.0.2        0 65015       0       0
    Active
```

**Note:** If a BGP session is cycling between *Connect* and *Active*, there is probably a problem with IP connectivity between peers, such as a physical link failure or IP routing problem.

### *OpenSent* State

The TCP connection between peers has been established and a BGP OPEN message has been sent. BGP is now waiting for an OPEN message from the other peer. When it receives an OPEN message, a KEEPALIVE message is immediately sent.

### *OpenConfirm* State

BGP has sent and received an OPEN message, has sent a KEEPALIVE message and is now waiting to receive a KEEPALIVE message.

**Note:** The transition from OpenSent and OpenConfirm is very fast.  It is almost impossible to see these states printed in CLI output, so no examples are shown here.

### *Established* State

BGP is now "up" and UPDATE messages can be exchanged.  KEEPALIVE messages are sent at a rate of 1/3 the *Hold Time* for the remainder of the session lifetime.  If any errors are detected or the *Hold* timer expires, a NOTIFICATION message is sent and BGP transitions back to *Idle*.  Instead of printing *Established* in the CLI output, the number of received and advertised prefixes is shown.  It is implied that BGP is in *Established* because no routing information could be exchanged if this were not true.

```
RS# bgp show summary
Local router ID is 192.168.0.1, Local AS number 65030
BGP Route Entries 3, Unique AS Paths 3
Unique Communities 0, Unique Extended Communities 0

Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes
Rcvd/Sent
--------        -    -- ------- -------      ------- ---------
--------
[Group Id: VVNet]
10.0.0.2        4 65015       5       5     0d0h2m0s
3/0
```

## Connection Collision Detection

It is entirely possible that two TCP sessions may be established between the same peers when BGP is in *OpenSent* and *OpenConfirm*. However, exactly one TCP session is required between two BGP speakers. A mechanism called *Connection Collision Detection* is implemented to decide which session to close and which one to keep using information from the OPEN message. The connection initiated from the router with the numerically highest router ID is kept in this situation, and the connection initiated from the router with the lowest router ID is closed.

---

[Home][Documentation]
[Index][Previous][Next]

# *BGP Fundamentals*

## Path Attributes

Path attributes are the set of parameters that describe the various path characteristics of a path to a destination IP prefix. They are used extensively in the route selection process to choose the best of multiple routes, and to build routing policies by matching and setting attributes. Each is represented as a TLV and sent with the associated NLRI in UPDATE messages. There are four categories of attributes which describe how to process and distribute the routing information to other BGP speakers.

- *Well-known mandatory*: must be present in an UPDATE message and must be implemented by all BGP speakers to ensure that all implementations support a standard set of attributes
- *Well-known discretionary*: may be present in UPDATE message and must be implemented by all BGP speakers, sometimes it does not make sense for a particular path attribute to be present
- *Optional transitive*: may be present in UPDATE message and may be implemented by a BGP speaker; passed on to other BGP speakers, even if it is not understood by the local BGP speaker
- *Optional non-transitive*: may be present in UPDATE message and may be implemented by a BGP speaker; ignored and not passed on to other BGP speakers, if it is not understood by the local BGP speaker

## Path Attribute Format

| 0 | 1 | 2 | 3 | 4 5 6 7 | |
|---|---|---|---|---|---|
| W/O | N/T | C/P | EL | Unused (4 bits) | Attribute Type Code (1 byte) |
| Attribute Length Code (variable) | | | | | Attribute Value (variable) |

The above diagram shows the format of each path attribute as it is sent in the Path Attributes field of an UPDATE message. The first few bits are used to describe each attribute's category, and the remaining fields contain the attribute's TLV.

- Bit 0: *Well-known* or *Optional*
- Bit 1: *Non-transitive* or *Transitive*
- Bit 2: *Complete* or *Partial*
  - *Complete*: attribute was passed along entire path
  - *Partial*: a router in the path did not implement an attribute, routing information may be lost (very rare in today's Internet as most routers support all path attributes)
- Bit 3: *Extended Length*, defines if the attribute length is 1 byte or 2 bytes
- Bits 4 – 7: unused and are set to 0
- *Attribute Type Code*: type code of the attribute
- *Attribute Length* : length of the attribute
- *Attribute Value*: data or value of the attribute

## Path Attribute Details

Many path attributes have been defined in RFCs and I-Ds, but not all of them have been implemented or are used today. This section will only describe and discuss the path attributes that are relevant to today's Internet routing in order to provide the necessary understanding required to implement complex and scalable networks. The IBGP Scalability section then introduces some more path attributes that are only used for route reflection or confederations.

***ORIGIN* (Type Code 1, Well-known Mandatory – RFC 1771)**

The *ORIGIN* defines the origin of the path information at the originating AS, as seen in the FIB at the originating router. It can have the following values.

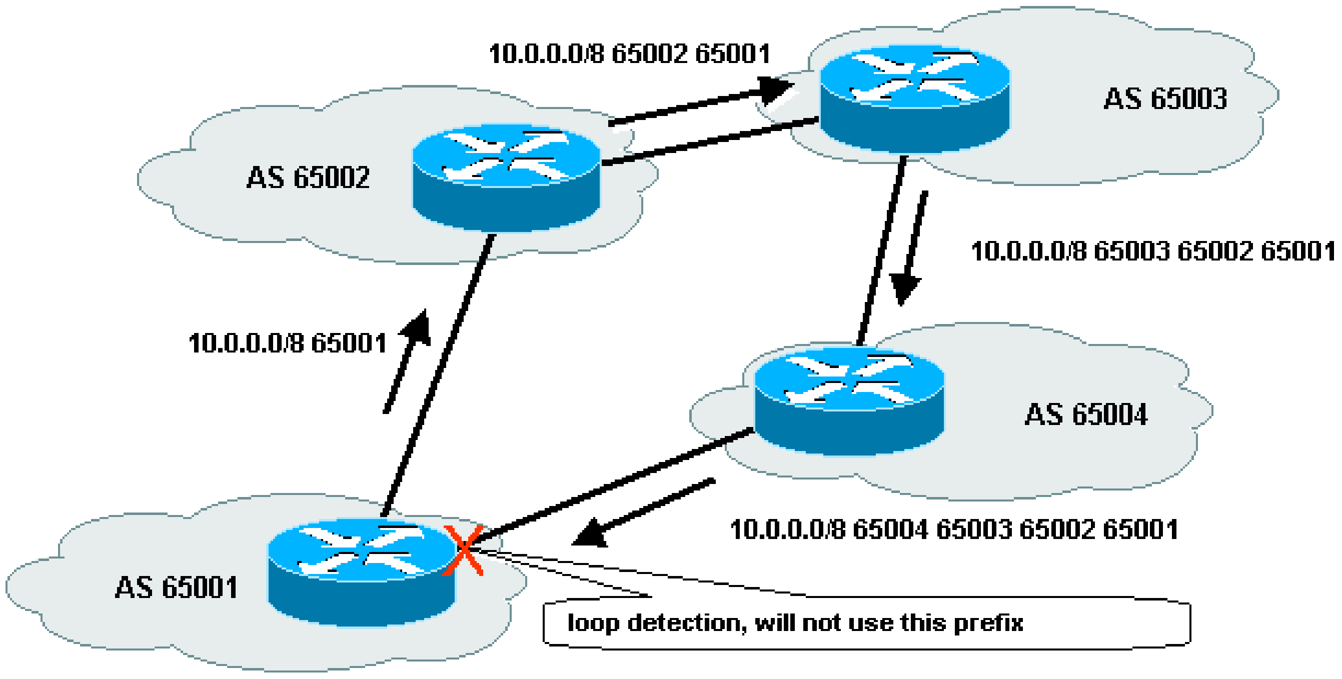
- IGP: the prefix was learned through an IGP; denoted with an "`i`" in CLI output
- EGP: the prefix was learned through the EGP protocol (should not see this today because no one runs EGP anymore); denoted with an "`e`" in CLI output
- Incomplete: the prefix was learned through redistribution or as an aggregate, it this does not indicate an error condition; denoted with a "`?`" in CLI output

***AS_PATH* (Type Code 2, Well-known Mandatory – RFC 1771)**

This attribute contains a list of the ASs the prefix has traversed. It also provides for loop detection by looking for local system's AS in the path. A BGP speaker's own AS is prepended to the AS path when advertised with EBGP, but not with IBGP. So, the AS that advertised the prefix will always be at the left side of an AS path, transit ASs will be in the middle, and the originating AS will be at the end of the AS path.

Two types of path segments are used to specify what the AS path means.

- *AS_SET* (path segment type 1): a unordered set of AS numbers, used to aggregate routes with different AS paths, and rarely seen in Internet routing table because this information is usually stripped off if it exists; denoted with “`[]`” in CLI output, for example: `[65001 65003 65002]`
- *AS_SEQUENCE* (path segment type 2): an ordered set of AS paths, from last advertised to origin AS; denoted as a string of ASs in CLI output, for example: `65001 65002 65003`



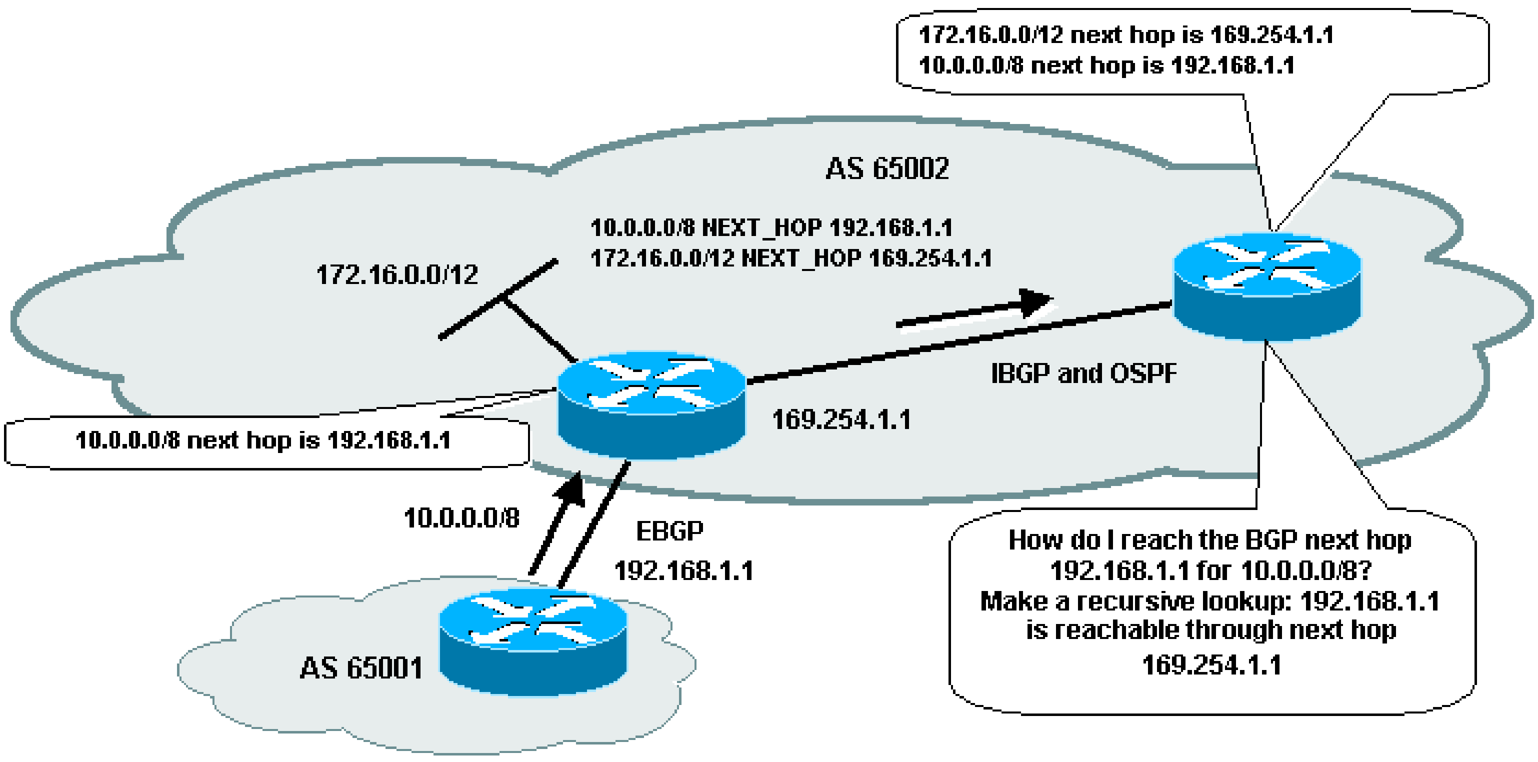
The above diagram illustrates the AS path attribute for the prefix 10.0.0.0/8, as it is advertised from AS 65001 to AS 65004. Each router prepends its AS when advertising the prefix to its EBGP peer. When the originating router detects its AS in the AS path, it discards that prefix.

## ***NEXT_HOP*** **(Type Code 3, Well-known Mandatory – RFC 1771)**

The *NEXT_HOP* attribute is the IP address of the router which should be used as the BGP next hop to the destination. The router makes a recursive lookup to find the BGP next hop in the routing table. The IGP usually has next hop route to BGP next hop, or it could be a static route or directly connected interface. This next hop must be reachable for consideration in route selection process because doesn’t make sense to install a route that isn’t reachable. The following rules apply for the BGP next hop, depending on the peering configuration.

- EBGP sets the next hop address to the IP address of the peer that advertised the prefix

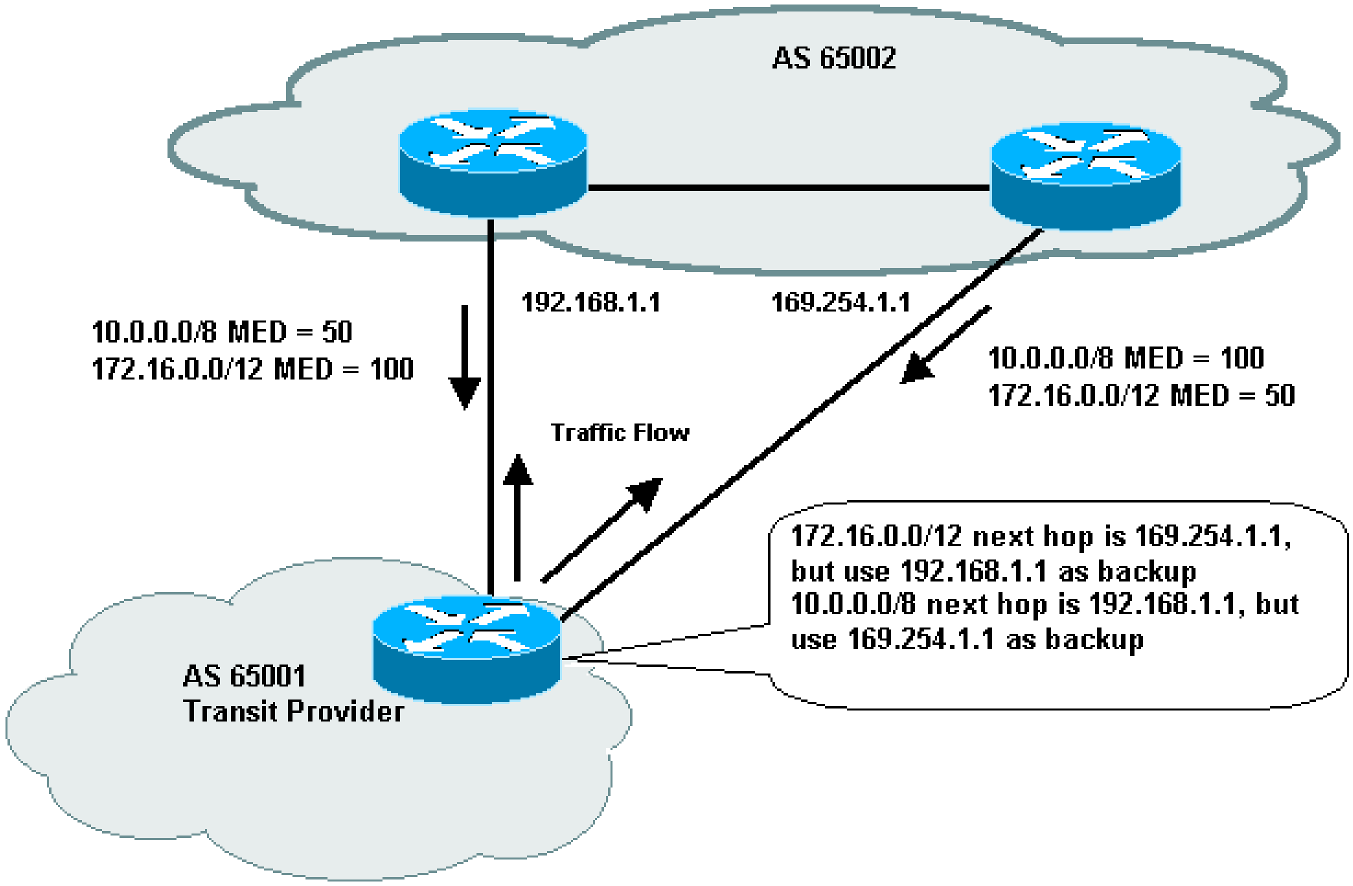
- IBGP sets the next hop address to the IP address of the peer that advertised the prefix for routes that originate internally
- IBGP passes the next hop unaltered for prefixes that are learned with EBGP


172.16.0.0/12 next hop is 169.254.1.1
10.0.0.0/8 next hop is 192.168.1.1
AS 65002
10.0.0.0/8 NEXT_HOP 192.168.1.1
172.16.0.0/12 NEXT_HOP 169.254.1.1
172.16.0.0/12
IBGP and OSPF
10.0.0.0/8 next hop is 192.168.1.1
169.254.1.1
10.0.0.0/8
EBGP
192.168.1.1
How do I reach the BGP next hop 192.168.1.1 for 10.0.0.0/8? Make a recursive lookup: 192.168.1.1 is reachable through next hop 169.254.1.1
AS 65001

An example of the *NEXT_HOP* attribute is shown in the above diagram. Each router must resolve the BGP next hop before the route can be used. Don't confuse the two next hops. The BGP next hop (the *NEXT_HOP* attribute) is carried by BGP, while the next hop address is used to resolve the BGP next hop from the FIB.

**_MULTI_EXIT_DISC_ or MED (Type Code 4, Optional Non-transitive – RFC 1771)**

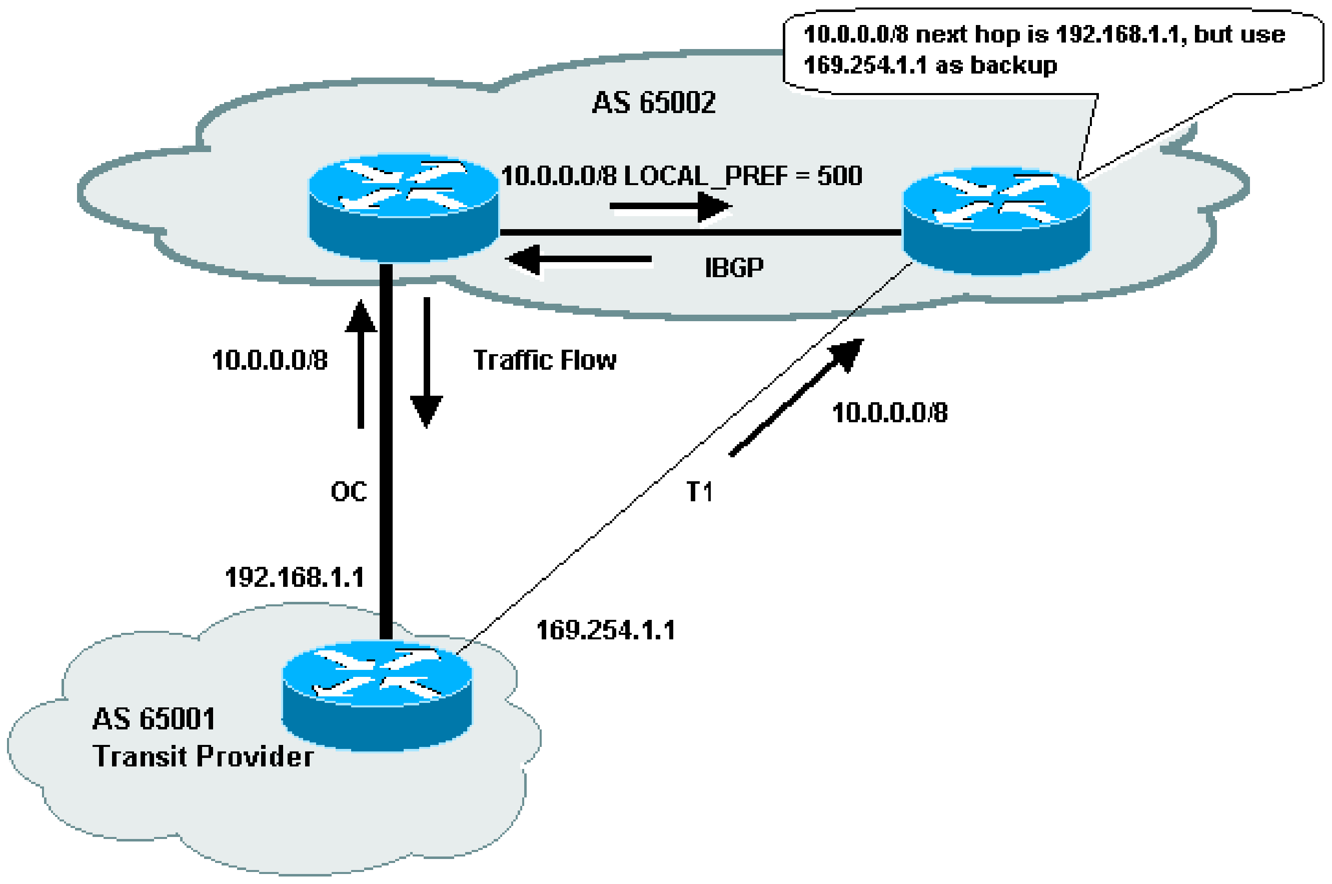
*MULTI_EXIT_DISC*riminator, also called a MED or simply the metric, is used to communicate a preferred path <u>into</u> an AS, it influences <u>ingress</u> traffic. MED values are locally significant to an AS and are set according to the local policy. Thus, comparing MEDs between different ASs is like comparing apple and oranges because you have no idea what they mean when they are compared. A MED of 100 can mean completely different things to different networks that have different MED policies. Although there is a CLI command that can enable MED comparison between different ASs, it should only be used with care and never enabled by default. For MEDs, a lower value means a higher preference and the values can range from 0 – 65535. Missing MEDs (-1) are treated as the highest preference.



The diagram above shows an example of how MEDs can be used to offer primitive load-distribution and backup paths. This is one of the most common applications of MEDs. The customer advertises prefixes with opposite low and high MEDs to the ISP, in order to direct ingress traffic to a particular router. When both links are up, the load is shared between the links. If a link fails, the providers' router still has a backup path through the router that is advertising the higher MED. The IGP metric can also be used as the MED to let the IGP dynamically direct ingress traffic to the best border router through BGP.

***LOCAL_PREF*** **(Type Code 5, Well-known Discretionary – RFC 1771)**

The *LOCAL_PREF*erence is a metric used to communicate a preferred path out of an AS, it influences egress traffic. This attribute is only significant within as AS. It is only permitted to be exchanged between IBGP peers, and may not be sent between EBGP peers. A higher value means a higher preference and can range from 0 to 65535. Its default value is 100, and can be changed with a CLI command.



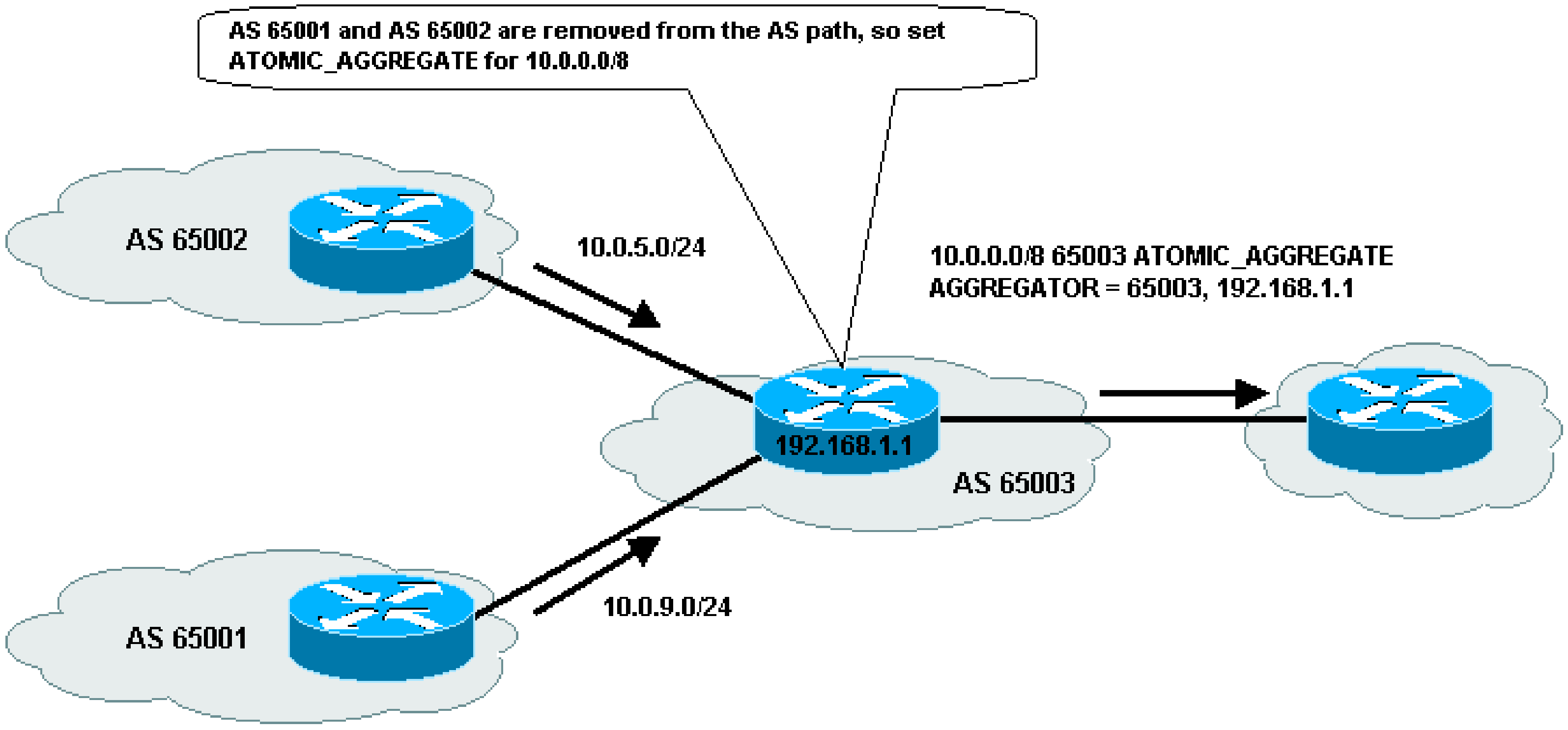
The diagram above shows an example of how local preference can be used to offer primitive load-distribution and backup paths. Local preference is commonly used to choose a particular exit from an AS. In the example, the customer is connected by an OC3 and a backup T1. All egress traffic from the customer's network should use the OC3, unless that circuit has failed. The customer sets a high local preference on routes received over the OC3 link, and keeps the default local preference of 100 on routes received over the T1 link. When both links are up, traffic is sent over the OC3 link according to the higher local preference. If the OC3 link fails, then traffic will be sent over the backup T1 link.

**ATOMIC_AGGREGATE (Type Code 6, Well-known Discretionary – RFC 1771)**

*ATOMIC_AGGREGATE* is set if a router advertises an aggregate causes path attribute information to be lost. If *ATOMIC_AGGREGATE* is set, then the prefix should not be disaggregated. See the example *AGGREGATOR* attribute example below.

***AGGREGATOR* (Type Code 7, Optional Transitive – RFC 1771)**

This attribute specifies the router ID and AS of the router that originated an aggregate prefix. It is useful for debugging.


AS 65001 and AS 65002 are removed from the AS path, so set ATOMIC_AGGREGATE for 10.0.0.0/8
AS 65002
10.0.5.0/24
10.0.0.0/8 65003 ATOMIC_AGGREGATE
AGGREGATOR = 65003, 192.168.1.1
192.168.1.1
AS 65003
AS 65001
10.0.9.0/24

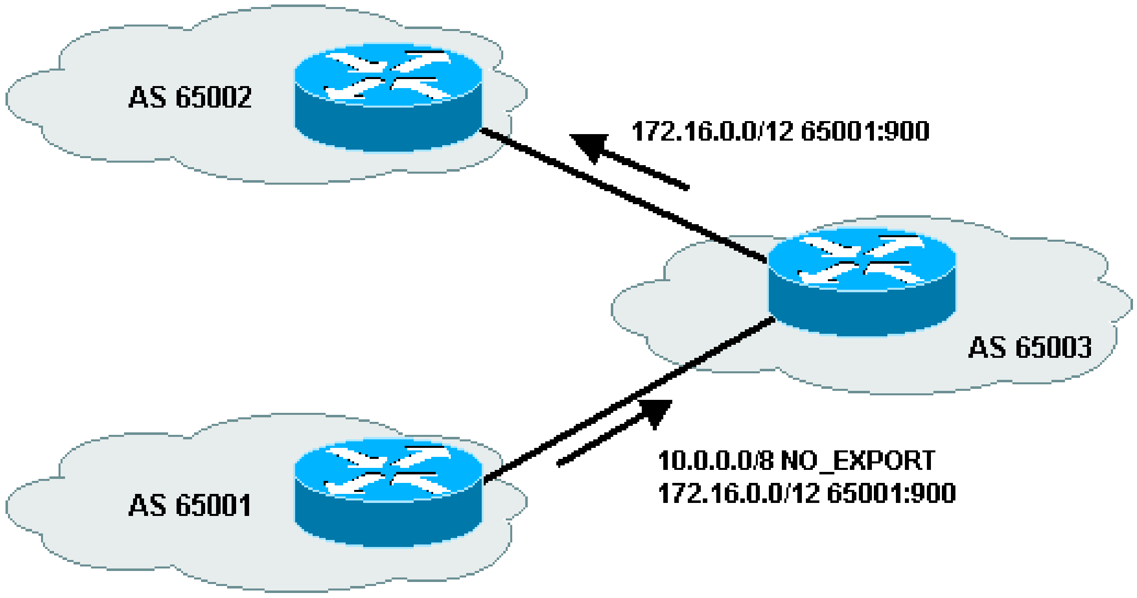
The diagram above illustrates the *ATOMIC_AGGREGATE* and *AGGREGATOR* attributes used together. 10.0.0.0/8 is allocated to AS 65003, which in turn allocates subnets of this network to its customers. Each customer advertises the /24 subnet with BGP. When AS 65003 advertises its aggregate /8, it causes a loss of AS path information because 65001 and 65002 will be removed from the aggregate's AS path. Additionally, the *AGGREGATOR* attribute is included to indicate which router originated the aggregate.

**_COMMUNITIES_ (Type Code 8, Optional Transitive – RFC 1991)**

*COMMUNITIES* provide a way to logically classify a prefix for use in policies by attaching an identifier that is significant within a network. For example, it can be used to convey meaning about where a prefix entered the network, or how a prefix should be advertised to different peers. Prefixes can have multiple communities, and policy decisions can be made on one or more of the communities. Or, the same community can be given to multiple prefixes.

In the CLI, communities are represented as two numbers separated by a "`:`", for example "`65001:500`" or "`65000:750`". Each number can have a range between 0 – 65535. The convention used is to set first number to the local AS, and the second number to an arbitrary value that is defined by the networks' administrative policy. This makes is possible to track which AS set a particular community. The communities "`0:0`" through "`0:65535`" and "`65535:0`" through "`65535:65535`" are reserved and cannot be used, so the CLI will not accept communities in this range. Some well-known communities from this reserved range have been defined for convenience and are understood by all routers.

- *NO_EXPORT* (65535:65281): should not be advertised to EBGP peer, useful for filtering advertisements at network boundaries; denoted with “`no-export`” in the CLI
- *NO_ADVERTISE* (65535:65282): should not be advertised to <u>any</u> peer; denoted with “`no-advertise`” in the CLI



The example in the above diagram shows how communities could be used to selectively suppress advertisements for a network to other peers. The well-known community “`no-export`” is given to 10.0.0.0/8. The router is AS 65003 receives this prefix and will forward traffic to this destination, but will not advertise that network to the router in AS 65002.

***EXTENDED COMMUNITIES* (Type Code 16, Optional Transitive – draft-ietf-idr-bgp-ext-communities-02.txt)**

*EXTENDED COMMUNITIES* share the same concepts as RFC 1991 *COMMUNITIES*, and function in the same way. They are implemented to allow a wider range of values, and have a more defined structure. The *EXTENDED COMMUNITIES* attribute is eight bytes in length, compared to four bytes for RFC 1991 communities and is defined as a two byte type followed by a six byte value. Each byte in the type determines the structure and meaning of the value. In the CLI, an extended community is represented as “`type:administrator:value`”. Several types of extended communities have been defined.

- *Route Target Community* (extended type 0x02): identifies a target for a prefix, transitive across AS boundaries; denoted in the CLI as “`target:AS:value`” or “`target:IP address:value`”, for example “`target:65000:500`” or “`target:192.168.0.1:1000`”
- *Route Origin Community* (extended type 0x03): identifies the origin of a prefix, transitive across AS boundaries; denoted in the CLI as “`origin:AS:value`” or “`origin:IP`

`address:value`", for example "`origin:65000:500`" or "`origin:192.168.0.1:1000`"

- *Link Bandwidth Community* (extended type 0x04): defines a metric for the link bandwidth between IBGP and EBGP peers, non-transitive across AS boundaries; will be supported by Riverstone in a future RapidOS version

Extended communities are not as widely implemented as RFC 1991 communities yet, and are usually only used in combination with MPLS VPNs.  So when people talk about communities today, they usually mean RFC 1991 communities.  Each of the communities can be mixed as needed in the CLI.

## Differences Between IBGP and EBGP

BGP can run in two modes that each has a very different behavior when advertising routing information.

- *EBGP*: external BGP runs between routers in different ASs
- *IBGP*: internal BGP runs between routers in the same AS

All transit (or "core") routers must have a consistent and complete view of all paths to external networks.  This is accomplished by configuring IBGP peering between all transit routers within an AS.  It allows for best egress path selection among multiple BGP routes that are learned from EBGP peers.  IBGP relies on IGP routes for peer address and next-hop address resolution.  There is a fundamentally behavior for each mode when advertising routes and path attributes which is as follows.

- *EBGP*: routes received from an EBGP peer can be advertised to EBGP and IBGP peers
- *IBGP*: routes received from an IBGP peer cannot be advertised to another IBGP peer but can be advertised to an EBGP peer

| Attribute | EBGP | IBGP |
|---|---|---|
| ORIGIN | Mandatory | Mandatory |
| AS_PATH | Mandatory | Mandatory |
| NEXT_HOP | Mandatory | Mandatory |
| MULTI_EXIT_DISC | Discretionary | Discretionary |
| LOCAL_PREF | Not Allowed | Required |
| ATOMIC_AGGREGATE | Discretionary | Discretionary |
| AGGREGATOR | Discretionary | Discretionary |
| COMMUNITY | Discretionary | Discretionary |

There is <u>no</u> loop detection in IBGP because if IBGP added its AS to the AS path it would look like a loop.  Thus only EBGP speakers prepend their AS when advertising routes.  Advertising IBGP routes to an IBGP peer could cause a routing loop or black hole.  The solution then is to fully mesh all IBGP routers in order to distribute routes, this means each IBGP router must have an IBGP session with all other BGP routers.  However, this presents some scalability problems.  Two mechanisms are supported to address the IBGP full mesh problem: route reflection and confederations.  Each is discussed in detail in the IBGP Scalability section.

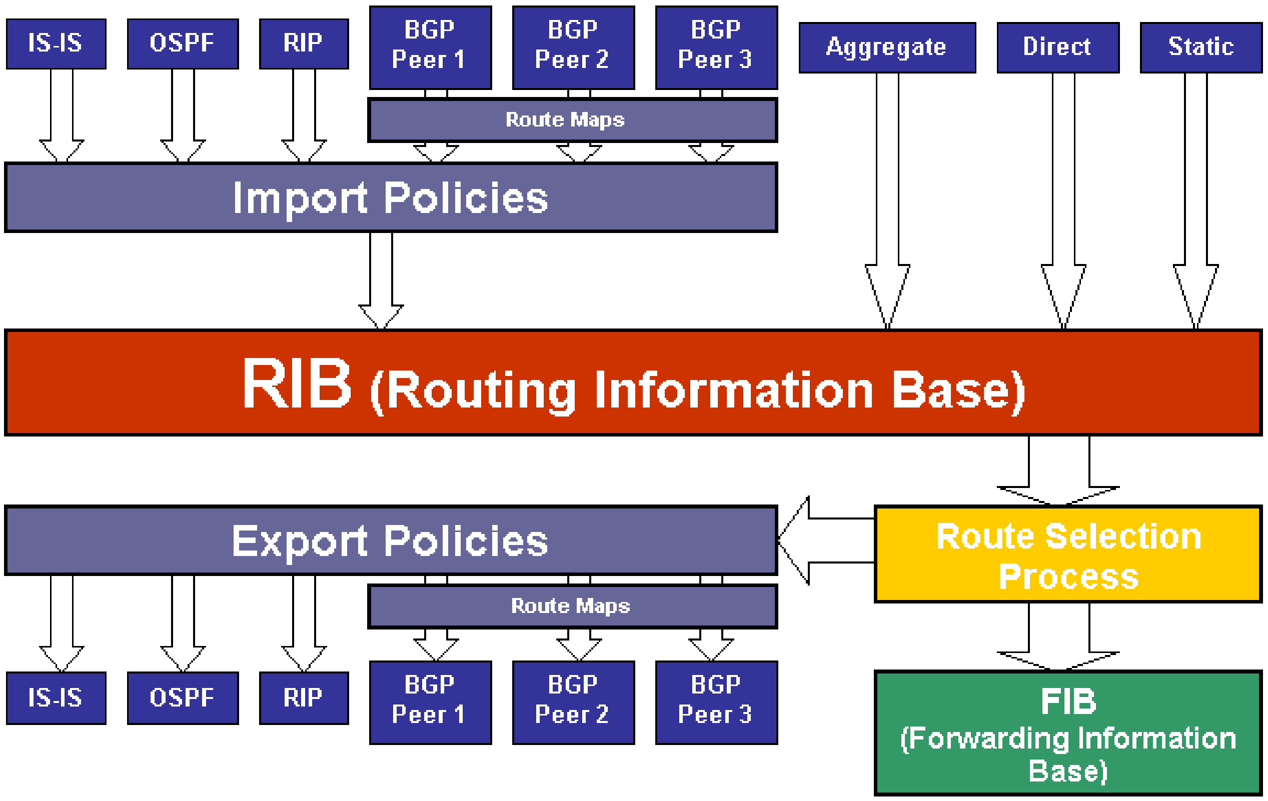
---

# RS Routing Model

An overview and discussion of the RS routing model and route selection process, which explains the RIB and FIB, and how the best route is selected to forward traffic.

## RIB and FIB Overview

Routes can be learned from many sources, for example direct, static, aggregate, IGP or EGP.  The Routing Information Base (RIB) holds all routes from each protocol source, which can be controlled with global import policies and route maps for BGP routes.  The Forwarding Information Base (FIB) holds the active (best) route and is used to forward traffic based on a longest prefix match.  The route selection process chooses the active route to be installed in the FIB from the available RIB routes.  FIB routes can then be exported to other protocols with global export policies, and route maps for BGP routes.



**BGP Specifics in the Routing Model**

RFC 1771 defines three RIBs that BGP uses to store routing information.

- Adj-RIBs-In: all routes received from a peer (`bgp show peer-host x.x.x.x all-received-routes`)
- Loc-RIB: all valid BGP routes (`bgp show routes all`)
- Adj-RIBs-Out: routes advertised to a peer (`bgp show peer-host x.x.x.x advertised routes`)

The Adj-RIBs-In and Loc-RIB are both stored in the router's RIB on a Riverstone router. Other vendors implement the three BGP RIBs in separate memory locations. Each is an acceptable approach, as these implementation details are left up to each vendor. Route maps enable additional policy control and allow you to match and modify path attributes on BGP routes that are imported or exported between the BGP RIBs.

## Route Selection Process

The route selection process for BGP is deterministic, it chooses best route in the same way every time according to a given set of rules. RIP, for example, is a non-deterministic routing protocol. The best BGP routes are chosen based on path attributes, other characteristics such as delay, bandwidth or latency do not matter for BGP route selection. RapidOS implements a single route selection process for all routes, the steps that test path attribute values are only used for BGP routes. If routes are equal at a given point in the selection process, then the next step is applied to break the tie until a best route is chosen.

It is important to understand that the router will only run the route selection process among RIB routes that have the same exact network and mask. There is no need to run route selection among different routes because there is no choice to make between them.

**Step 1: Active Route**

- If the next hop address is not accessible, do not consider the route.

For BGP routes the router makes a recursive lookup to find the next hop in the routing table, which is usually carried in the IGP.

**Step 2: Configured Policy**

- Consider the route with smallest *preference*, as defined by the route preferences. Ties are broken by the smallest *preference2* value.

Each protocol has a numerical preference between 0 and 255. These preferences are global to the router and defaults can be changed for each protocol with CLI commands. BGP route preferences can be changed globally for all BGP routes, or for each peer with an import policy. The default route preference for each protocol can be found with the command "`ip-router show route-preferences`" and are shown in the table below. Preference2 defaults to 0 for all protocols.

| Type | Preference | Type | Preference |
|---|---|---|---|
| Direct | 0 | Router Discovery | 55 |
| Direct Aggregate | 0 | RIP | 100 |
| Static | 5 | Aggregate | 130 |
| OSPF | 10 | OSPF ASE | 150 |
| IS-IS L1 | 15 | IS-IS L1 External | 160 |
| IS-IS L2 | 18 | IS-IS L2 External | 165 |
| Default | 20 | BGP | 170 |

If the route is not a BGP route, go to step 8.

**Step 3: Local Preference (BGP only)**

- If the BGP preferences match (*preference* and *preference2*), prefer the route with the highest (best) BGP local preference.

The local preference path attribute is not sent by EBGP peers, and defaults to 100 if none is present.

**Step 4: Shortest AS Path (BGP only)**

- If the routes have the same BGP local preference, prefer the route with the fewest Autonomous Systems listed in its AS path.

**Step 5: Origin IGP < EGP < Incomplete (BGP only)**

- If routes have the same Autonomous System path length, prefer the route with the lowest origin code. IGP is preferred over an origin of EGP. Least preferred is an Autonomous System path that is incomplete.

**Step 6: MED (BGP only)**

- If origin codes are the same, prefer the lowest (best) Multi-Exit Discriminator. MEDs are only compared between routes that were received from the same neighboring Autonomous System unless "`bgp set compare-med always`" is configured. A missing metric, indicated by a value of -1, is treated as the lowest (best) MED.

**Step 7: Shortest IGP Distance (BGP only)**

- If the MEDs are the same, prefer the route whose next hop IP address is closer with respect to the IGP distance.

**Step 8: Source IGP < EBGP < IBGP**

- If the IGP distances are the same, first prefer the strictly interior route, then the strictly exterior route, then the exterior route learned from an interior session.

This means to prefer the IGP over EGP, and EBGP over IBGP. The IGP is considered more trusted than an EGP, because it does not contain routes external to the network.

**Multipath Routing**

Multipath routing is enabled by default and supports up to four equal cost paths for each protocol. At this point multiple active routes will be installed in the FIB. If multipath is disabled, then step 9 will be used to break a tie in a deterministic way.

**Step 9: Lowest Router ID (tie breaker only)**

- For BGP: If the sources are the same, prefer the route with the numerically lowest BGP router ID.
- For all other protocols: If the sources are the same, prefer the route whose next hop IP address is numerically lowest.

**BGP Route Selection Example**

With BGP multipath enabled, the BGP routes are displayed as follows.

```
RS# bgp show routes all
Local router ID is 192.0.2.1
Status codes: > - best, * - valid, i - internal, t - stale
              s - suppressed, d - damped
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network            Next Hop        Metric LocPrf Path
    -------            --------        ------ ------ ----
*>  172.16/12          10.0.0.2                  100 (65030) 65015 ?
*>  172.16/12          10.0.0.6                  100 (65030) 65100 ?

RS# ip show routes

Destination          Gateway              Owner     Netif
-----------          -------              -----     -----
10.0.0.0/30          directly connected   -         VVNet
10.0.0.4/30          directly connected   -         Crest
172.16.0.0/12        10.0.0.2             BGP       VVNet
                     10.0.0.6             BGP       Crest
```

If BGP multipath is disabled, then the output will change to the following. In this case, all path attributes are equal, and the router ID was used to break the tie between routes.

```
RS# bgp show routes all
Local router ID is 192.0.2.1
Status codes: > - best, * - valid, i - internal, t - stale
              s - suppressed, d - damped
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network            Next Hop        Metric LocPrf Path
    -------            --------        ------ ------ ----
*>  172.16/12          10.0.0.2                  100 (65030) 65015 ?
*   172.16/12          10.0.0.6                  100 (65030) 65100 ?

RS# bgp show routes 172.16/12
Local router ID is 192.0.2.1
BGP routing table entry for 172.16/12
Path: Best
Source: 10.0.0.2
Router Id : 10.0.0.2
Advertised to Tasks:
 3-KRT 4-BGP_65100.10.0.0.6+1032
Flap statistics: None
Local AS: 65030   Peer AS: 65015   Age: 3:43
NextHop: 10.0.0.2          MED:  -1    Local Preference: 100
AS Path: (65030) 65015 ?

BGP routing table entry for 172.16/12
Path: Not Best
Source: 10.0.0.6
Router Id : 192.168.0.34
Advertised to Tasks: None
Flap statistics: None
Local AS: 65030   Peer AS: 65100   Age: 13:20
NextHop: 10.0.0.6          MED:  -1    Local Preference: 100
AS Path: (65030) 65100 ?
```

# *Basic BGP Configuration Tasks*

This section covers examples and tips on how to configure BGP that include setting global router parameters, followed by basic IBGP and EBGP peering.

## Configuration Basics

Before BGP can be configured, two global router settings must be configured first: the router ID and autonomous system.

The router ID is an IP address that uniquely identifies the router, and is usually set to be the same as the router's loopback address.

Syntax:
```
RS(config)# ip-router global set router-id <IP address>
```

Example:
```
RS(config)# ip-router global set router-id 192.0.2.1
```

The router's autonomous system is set as follows.

Syntax:
```
RS(config)# ip-router global set autonomous-system <AS>
```

Example:
```
RS(config)# ip-router global set autonomous-system 65030
```

All BGP peers must be configured as part of a peer group in RapidOS. Peer groups allow logical grouping of the same type of peer for ease of configuration and scalability. All settings applied to a peer group are global to the group and are inherited by each peer. This allows common options or policies to be applied to all peers with a single configuration line. Scalability is improved when grouping peers with the same routing policies together, for example IBGP peers. Since UPDATE messages for peer groups are composed once and then simply replicated for each peer in the group, CPU cycles are saved and routing information is distributed faster. Peer groups can contain one or more peers and are configured with the following command.

Syntax:
```
RS(config)# bgp create peer-group <name> autonomous-system <remote AS>
```

The group name is a simply a descriptive name for the group, thus good group naming conventions will help with readability and troubleshooting. The remote AS is the AS number of the other peer. If this matches the router's AS, then the group will be an IBGP group and if it does not match then it will be an EBGP group. After creating the peer group, peers can be added to the group.

Syntax:
```
RS(config)# bgp add peer-host <IP address> group <name>
```

The last step is to start the BGP protocol, no BGP processing is done without this command**.**

Syntax:
```
RS(config)# bgp start
```

## Configuring Basic IBGP Peering

IBGP should always be configured between router loopback addresses. The loopback address is the most stable interface, it never goes down and is not susceptible to physical failures. Also, there is often more than one internal path to the loopback address in the network. So, this configuration will keep the IBGP session stable by using IGP to route around link failures without tearing down the IBGP session.

Example:
```
RS(config)# bgp create peer-group IBGP autonomous-system 65030
RS(config)# bgp add peer-host 192.0.2.2 group IBGP
```

The source IP address must be set to the loopback address when peering between loopback interfaces, because the IP packet's default source address is the address of the egress IP interface.

Syntax:
```
RS(config)# bgp set peer-group <name> local-address <IP address>
```

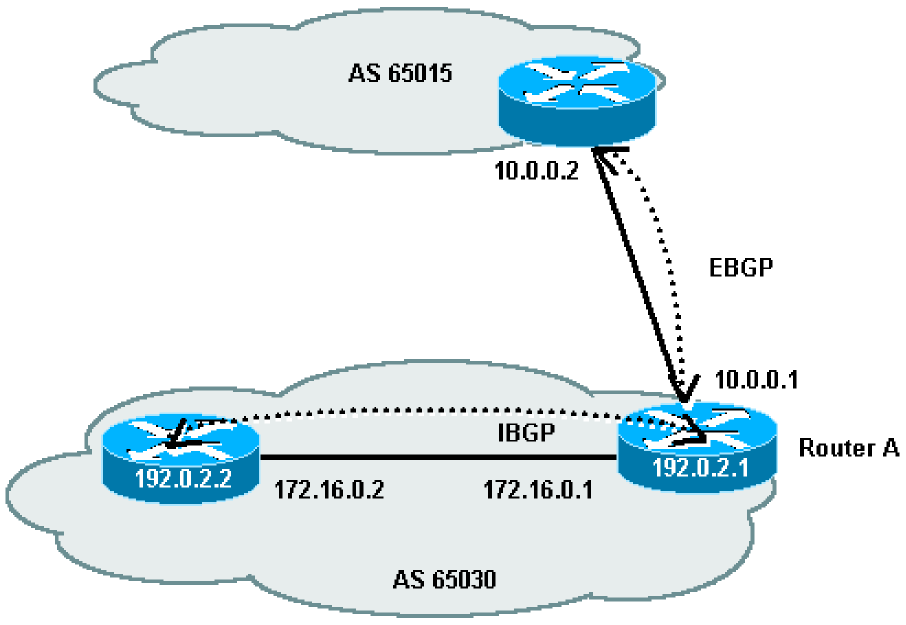
Example:
```
RS(config)# bgp set peer-group IBGP local-address 192.0.2.1
```

## Configuring Basic EBGP Peering

EBGP should be configured between interface addresses in most cases, since there is usually only one link between external routers. If a link fails, the peering session will be terminated immediately, without waiting the for *Hold* timer to expire. Another added benefit is that this configuration will not rely on the IGP for reaching the peer address, so instabilities in the IGP will not affect EBGP peering or cause flapping.

Example:
```
RS(config)# bgp create peer-group VVNet autonomous-system 65015
RS(config)# bgp add peer-host 10.0.0.2 group VVNet
```

## Configuration Example



A complete configuration example for IBGP and EBGP is provided here as a reference for Router A in the above diagram.

```
!
! create IP interfaces
!
interface create ip VVNet address-netmask 10.0.0.1/30 port so.1.1
interface create ip internal address-netmask 172.16.0.1/29 port gi.2.1
interface add ip lo0 address-netmask 192.0.2.1/32
!
! global settings
!
ip-router global set autonomous-system 65030
ip-router global set router-id 192.0.2.1
!
! IGP is OSPF
!
ospf create area backbone
ospf add interface internal to-area backbone
ospf add stub-host 192.0.2.1 to-area backbone cost 10
ospf start
!
! BGP
!
! create peer groups
bgp create peer-group IBGP autonomous-system 65030
bgp create peer-group VVNet autonomous-system 65015
!
! add peers to the groups
!
bgp add peer-host 10.0.0.2 group VVNet
bgp add peer-host 192.0.2.2 group IBGP
!
! set group or peer options
!
bgp set peer-group IBGP local-address 192.0.2.1
!
! start BGP
!
```

```
    bgp start
```

## Peer Options Negotiation

BGP uses its OPEN messages to negotiate several options between peers. Each peering session may negotiate different values, depending on the peer's capabilities and default settings.

### *Hold Time*

The *Hold Time* must be agreed upon between two peers and is negotiated when OPEN messages are exchanged. The smaller value of the two peer's *Hold Time* is used for each BGP session, they can be different for different BGP sessions. RapidOS defaults to 180 seconds, and if desired this value can be changed with the following command.

Syntax:
```
RS(config)# bgp set peer-group <name> hold-time <seconds>
RS(config)# bgp set peer-host <IP address> hold-time <seconds>
```

### Capabilities Advertisement - RFC 2842

BGP speakers can advertise and negotiate the use of capabilities in OPEN messages between peers. Capabilities are extensions to the BGP protocol that enable additional features. The capabilities advertisement is described in RFC 2842, and various RFCs and I-Ds define additional capabilities. Its negotiation process is very simple and deterministic, a BGP speaker simply may not use a capability that was not advertised by its peer. The negotiation ensures that the peers do not use any features that are not mutually supported. The following example capabilities are supported in RapidOS, and are described in detail in later sections.

- IPv4 unicast (RFC 2858): exchanges IPv4 unicast routes
- IPv4 multicast (RFC 2858): exchanges IPv4 multicast routes
- Route Refresh (RFC 2918): request a retransmission of a peer's routes without resetting session
- Graceful Restart (draft-ietf-idr-restart-02.txt): restart BGP without causing a route flap

## Using BGP CLI Commands

In this section the most commonly used BGP CLI show commands are discussed with example output. For a complete list of CLI and configuration commands, please refer to the product manuals.

**Show a Summary of BGP Information:** `bgp show summary`

Use the following command to show summary information about BGP, the peer groups and each peer.

```
    RS# bgp show summary
    Local router ID is 192.0.2.1, Local AS number 65030
    BGP Route Entries 100353, Unique AS Paths 23648
    Unique Communities 32, Unique Extended Communities 0

    Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes Rcvd/Sent
    --------        -    -- ------- -------      ------- ------------------
    [Group Id: IBGP]
    192.0.2.2       4 65030      24  100376   0d2h31m38s    0/100353
    [Group Id: VVNet]
    10.0.0.2        4 65015  100377      25   0d2h26m34s    100353/0
    BGP summary, 2 groups, 2 peers
```

**Show Global BGP Information:** `bgp show globals`

This command shows the router's global BGP settings, including default configuration values.

```
RS# bgp show globals
Router ID                 : 192.0.2.1
Autonomous System         : 65001
BGP default preference    : 170
BGP default localpref     : 100
BGP default metric        : -1
Action on a bad aspath    : Reset session
Comparison of MED         : Only for same AS
```

**Show Neighbor Details:** `bgp show neighbor`

To show detailed information about a BGP peer, use the following command. State information, configuration parameters, message counters and negotiated capabilities are printed in the command's output.

```
RS# bgp show neighbor 10.0.0.2
Peer: 10.0.0.2+32783    Local: 10.0.0.1+179     Type: External   remote AS 65015
State: Established      Flags: <GenDefault>
Last State: OpenConfirm Last Event: RecvKeepAlive       Last Error: None
Options: <>
Configured parameters :
Used parameters :
Peer Version: 4 Peer ID: 10.0.0.2       Local ID: 192.0.2.1     Active Holdtime: 180
Uptime 0d0h10m7s
Last traffic (seconds): Received 7      Sent 7  Checked 7
Input messages :Total        13    Updates         1    Octets       298
Output messages:Total        14    Updates         1    Octets       325
count of sent routes               3
count of recvd routes              1
count of route refresh recvd       0
count of route refresh sent        0
Supported capabilities
    V4 Multi attributes;V4 Uni;Route Refresh;
```

**Show BGP Routes:** `bgp show routes`

This command can be used to show all BGP routes in a summary form if the keyword "`all`" is specified, or can be used to show detailed information about a particular prefix if a prefix is specified.

```
RS# bgp show routes all
Local router ID is 192.0.2.1
Status codes: > - best, * - valid, i - internal, t - stale
            s - suppressed, d - damped
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network            Next Hop        Metric LocPrf Path
    -------            --------        ------ ------ ----
*>  169.254/16         10.0.0.2                  100 (65030) 65015 i


RS# bgp show routes 169.254.0.0/16
Local router ID is 192.0.2.1
BGP routing table entry for 169.254/16
```

```
    Path: Best
    Source: 10.0.0.2
    Router Id : 10.0.0.2
    Advertised to(Tasks):
    3-KRT
    Flap statistics: None
    Local AS: 65030   Peer AS: 65015   Age: 1:15
    NextHop: 10.0.0.2          MED:  -1    Local Preference: 100
    AS Path: (65030) 65015 i
```

**Show A Peer's Advertised Routes:** `bgp show peer-host` ***<IP address>*** `advertised-routes`

To show a peer's advertised routes, use the following command.

```
    RS# bgp show peer-host 10.0.0.2 advertised-routes
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                s - suppressed, d - damped
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network            Next Hop        Metric LocPrf Path
        -------            --------        ------ ------ ----
    *>  172.16/29          10.0.0.1                      65030 i
```

**Show A Peer's Received Routes:** `bgp show peer-host` ***<IP address>*** `received-routes`

Similarly, to show a peer's received and used routes, use the following command. This command shows the path attributes that received before any policies are applied. To show all received routes (used and unused), use the command "`bgp show peer-host` <***IP address***> `all-received-routes`" instead.

```
    RS# bgp show peer-host 10.0.0.2 received-routes
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                    s - suppressed, d - damped
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network            Next Hop        Metric LocPrf Path
        -------            --------        ------ ------ ----
    *>  169.254/16         10.0.0.2                      (65030) 65015 i
```

## Default Configuration Comparison with Cisco

"`ip bgp-community new-format`" is enabled by default (uses *n:n* format). The old format is not supported.
"`neighbor fast-external-failover`" is enabled by default (peering session terminates immediately if the link goes down).
"`neighbor send-community`" is enabled by default and can be changed with a route map (sends community path attribute).
"`neighbor soft-reconfiguration inbound`" is enabled by default (stores an unmodified copy of received prefixes so policy changes can be applied without a hard clear).
"`no auto-summary`" is enabled by default (no summary routes are generated, unless configured).
"`no synchronization`" is enabled by default (disables IGP to EGP synchronization).

# Routing Policies

This section defines and explains the many different routing policies available on the RS platform. Global routing policies, route maps and policy building blocks such as AS path regular expressions and AS path lists, community lists and prefix lists are covered in detail. A discussion on route aggregation and redistribution explains different ways of advertising routes with BGP.

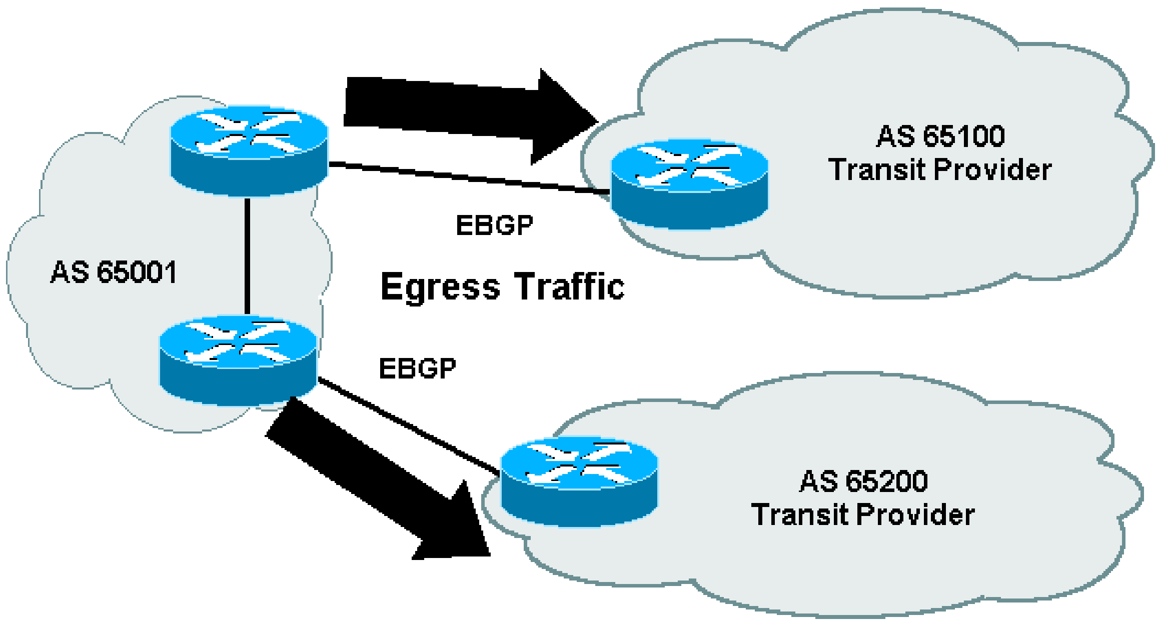
## Introduction to Policies

Routing policies enable route classification for importing and exporting routes. The goal of routing policies is to control traffic flow in a way that the network operator wants, which sometimes may not be the way that traffic will flow if directed by a routing protocol. Policies give very powerful control to the operator to change metrics and path attributes, deny or prefer certain routes, or any other action needed to control the RIB and FIB routes. Two types of routing policies are supported in RapidOS:

- Global routing policies: IP router policies that can be used for all protocol sources (discussed in this section)
- BGP policies: route maps that are only used for BGP routes (discussed in the Route Maps section)

### Routing Policies for Egress Traffic

Egress traffic is traffic that leaves your network, for example traffic that flows from AS 65001 to upstream transit ISPs.

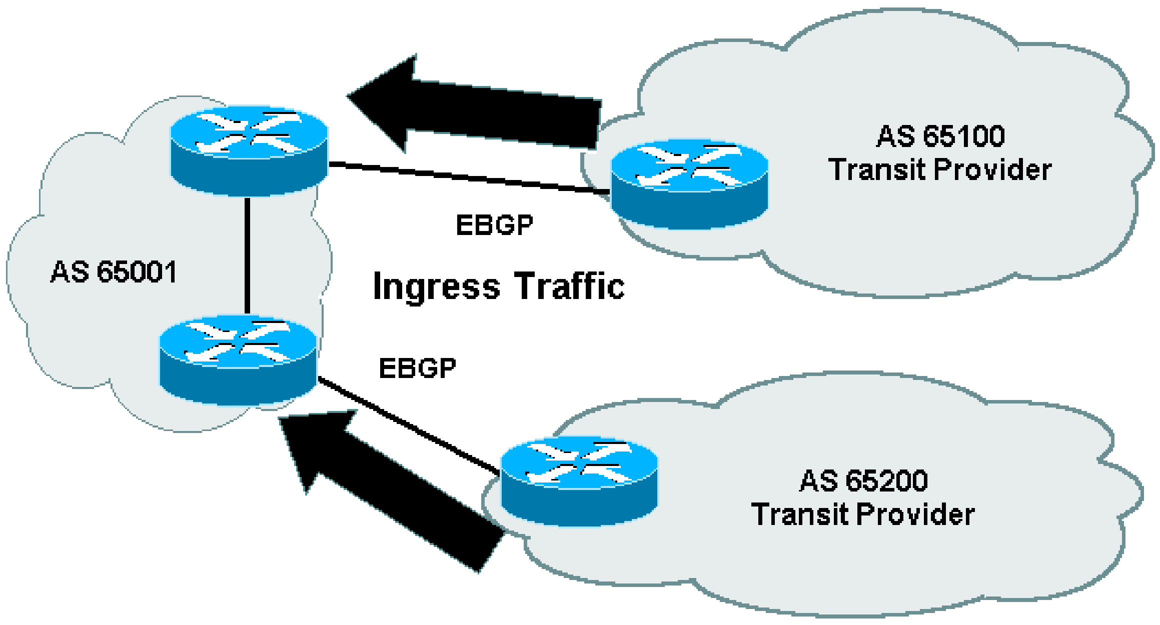


How traffic leaves your network is determined by many factors, such as:

- Your role in the peering relationship, ISP customers and Transit ISPs have different business goals that affect their traffic pattern
- Prefixes that are accepted from EBGP peers and are evaluated by the route selection process, more prefixes received result in more feasible routes
- Local preference tuning, which can be used to prefer routes from one EBGP peer over another
- AS path length on the received prefixes, often the prefix with the shorted AS path is installed as the best route

**Routing Policies for Ingress Traffic**

Ingress traffic is traffic that enters your network, for example traffic that flows to AS 65001 from upstream transit ISPs.



Which traffic enters your network is determined by many factors, such as:

- Your role in the peering relationship, ISP customers and Transit ISPs have different business goals that affect their traffic pattern
- Prefixes that you advertise to EBGP peers, more prefixes advertised result in more destinations to your network
- MEDs, which can be used to prefer paths intp your network (usually from the same provider)
- AS path length on the advertised prefixes, the prefix with the shortest AS path is preferred in most cases
- Other parties local preference, which can prefer paths into or through your network over others

## Global Routing Policies

By default, the router will import all BGP routes from all peers into the RIB, and export the best BGP route to all peers. This behavior can be change with policies. When a policy is configured, the default behavior stops for that particular peer or AS. So, if you want to redistribute static routes and BGP routes to a peer, you would configure one policy to export static routes, and one to export BGP routes. If you only wanted to export BGP routes, the default behavior will do this.

IP router policies are global to the router, and are used for protocol-specific actions such as redistribution into another protocol. These policies share some common options, which can be configured to permit or deny all or particular routes from a protocol.

- *exact***:** the mask of the routes must match the supplied mask exactly, used to match a network but not subnets or hosts of that network

- *refines***:** the mask of the routes must be more specific (i.e. longer) than the supplied mask, used to match subnets
- *between***:** the mask of the routes must be as or more specific (i.e., as long as longer) than the lower limit and no more specific (i.e. as long as or shorter) than the upper limit
- *restrict*: deny action for matching routes

The default is both *exact* and *refines*.

In most cases, you only want to redistribute static or aggregate routes into BGP.  It is generally a very bad idea to redistribute IGP routes into BGP.  Each redistribution command must specify a source and destination protocol.  For BGP the source AS may be specified, and target AS must be specified which gives you very granular and flexible control.  Be careful though, redistribution commands redistribute all routes from a protocol unless restricted in the policy.

Syntax:
```
RS(config)# ip-router policy redistribute from-proto <source> to-proto <destination> source-as
<source AS> target-as <target AS> [options]
```

Example, redistribute aggregate into BGP for all peers:
```
RS(config)# ip-router policy redistribute from-proto aggregate to-proto bgp target-as all
```

Example, only redistribute aggregate into BGP for AS 65015:
```
RS(config)# ip-router policy redistribute from-proto aggregate to-proto bgp target-as 65015
```

Example, do not redistribute BGP routes between AS 65000 and AS 65015:
```
RS(config)# ip-router policy redistribute from-proto bgp to-proto bgp source-as 65000 target-as
65015 restrict
```

The options such as *exact*, *refines*, *between* and *restrict* can be specified here for more granular control.

## Route Aggregation and Advertisement

Route aggregation generates a summary route, given the presence of a more-specific route.  It is used extensively to reduce the routing table size, internally and on the Internet by hiding unnecessary routing information.  Without aggregation the Internet routing table would have grown very big and out of control.  For example, many operators want to generate a BGP route given the presence of one or more subnets of that network learned via an IGP.  Aggregation is also a powerful tool to control routing by advertising aggregates intelligently to different providers with different path attributes in order to gain redundancy and load-sharing.

Aggregation must be explicitly configured, no aggregation is enabled by default because it depends entirely on what the operator intends to do.  Next, we will look at some examples that show two different ways to advertise aggregates.

### Route Aggregation Example 1

In this case, we want to advertise an aggregate for a 169.254.0.0/16. We have allocated several subnets of this network which are in the IGP or are directly connected to the router.  We want the aggregate to be dynamically withdrawn from BGP if there are no more-specific routes, for example if the IGP fails.  First, an aggregate route must be created.

Syntax:
```
RS(config)# ip-router policy summarize route <route/mask>
```

Example:

```
RS(config)# ip-router policy summarize route 169.254.0.0/16
```

Then, the aggregate route must be redistributed into BGP so it can be advertised to peers.

Syntax:

```
RS(config)# ip-router policy redistribute from-proto <source> to-proto <destination> target-as
<target AS>
```

Example:

```
RS(config)# ip-router policy redistribute from-proto aggregate to-proto bgp target-as all
```

The CLI output below shows the aggregate /16 route in the FIB, which is the advertised to the peer 10.0.0.2.

```
RS# ip show routes

Destination          Gateway               Owner     Netif
-----------          -------               -----     -----
169.254.0.0/24       directly connected    -         internal
169.254.0.0/16       127.0.0.1             Aggregate lo0

RS# bgp show peer-host 10.0.0.2 advertised-routes
Local router ID is 192.0.2.1
Status codes: > - best, * - valid, i - internal, t - stale
    s - suppressed, d - damped
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network            Next Hop        Metric LocPrf Path
    -------            --------        ------ ------ ----
*>  169.254/16         10.0.0.1                      65030 i
```

**Route Aggregation Example 2**

In this example, we want to statically advertise an aggregate for a 172.16.0.0/12 even if there is no route with a longer prefix.  This decouples the IGP from BGP, allowing the routes to be advertised with BGP even if the IGP is unstable.  Advertising BGP routes this way will hide internal instabilities or changes from EBGP peers and will avoid route flapping.

A simple command can be used to advertise routes with BGP (introduced in ROS 9.0.0.0).  There must be a route with the same or longer mask in the FIB in order to advertise the route.

Syntax:

```
RS(config)# bgp advertise network <network/mask>
```

Example:

```
RS(config)# bgp advertise network 192.168.0.0/16
```

In addition, the "`no-aggregate`" option can be specified to only allow routes with the same mask in the FIB to advertise the route.

A static route with the "`blackhole`" option and a high preference is created first to allow the route to be exported, with the added benefit that traffic for networks with no more-specific routes is discarded without sending ICMP messages.  The high preference ensures that a route from another protocol would be used first if it were in the RIB.  This configuration is akin to a Null0 route on a Cisco router.

Syntax:

```
RS(config)# ip add route <network/mask> gateway 127.0.0.1 blackhole preference 255
RS(config)# bgp advertise network <network/mask>
```

Example:

```
RS(config)# ip add route 172.16.0.0/12 gateway 127.0.0.1 blackhole preference 255
RS(config)# bgp advertise network 172.16.0.0/12
```

The CLI output below shows the aggregate /12 route in the RIB, which is the advertised to the peer 10.0.0.2.

```
    RS# ip-router show route 172.16/12
    mask 255.240
    entries 2    announce 1
            Aggregate Depth: 1
    TSI:
            BGP 10.0.0.2 (External AS 65015) no metrics
            Instability Histories:
            *        Static    Preference: 255
    *NextHop: 127.0.0.1          Interface: 127.0.0.1(lo)
    State: <Int Gateway Blackhole Unicast ActiveU>
    Age: 8:09    Metric: 0    Metric2: 0    Tag: 0
    Task: RT
    Announcement bits(2):
            3-KRT 4-BGP_65015.10.0.0.2+179 5-BGP_65038.192.168.0.38+179
    AS Path: Incomplete (Id 1)
                Aggregate    Preference:  -1
    State: <Int Hidden Gateway Reject>
    Age: 6:55    Metric: 0    Metric2: 0    Tag: 0
    Task: Aggregate
    No announcement bits set
            Flags:

    RS# bgp show peer-host 10.0.0.2 advertised-routes
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
        s - suppressed, d - damped
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network            Next Hop        Metric LocPrf Path
        -------            --------        ------ ------ ----
      *>  172.16/12          10.0.0.1                    65030 ?
```

---

# Routing Policies

## Regular Expressions

Regular expressions are a notation for describing patterns in a string of text. They are used to specify the rules for a set of strings that you want to match in a search. Sometimes they are called “regexp” or “regex” for short, these terms all mean the same thing.

For BGP, we use regular expressions to search AS paths to see if there is a match for a particular pattern. The regular expressions can be used in CLI commands to search for a particular AS path, or as building blocks to build complex policies. It is quite easy to advertise a few or all routes, but very hard to write a policy to match a particular 100 from a full routing table. Regular expressions allow a network operator to solve this problem. You can then ask questions such as “Is there a match for this AS pattern anywhere in the AS path?” and apply a policy to the matching prefixes.

Regular expressions are extremely powerful and flexible with binary logic, and are evaluated from left to right in sequence. Numbers denote a literal numeral, in our case, an AS number. Special characters are used to denote position or an operation within a string. These two components can be used together to form a regular expression containing any combination of literals and special characters. Riverstone implements a subset of POSIX regular expressions, which match neatly with Juniper’s syntax, and match Cisco’s syntax with some minor translation.

### Regular Expression Characters

The following characters can be used to build regular expressions.

- ***0, 1, 2, 3, 4, 5, 6, 7, 8, 9***: numbers are literals, used in any combination to represent an AS number
- **.** *(period* or *dot)*: represents any valid AS number.
- *(asterisks)*: represents zero or more of the preceding AS number, often used in conjunction with the period in order to wild card a sequence of AS numbers
- ***?*** *(question mark)*: represents zero or one of the preceding AS number
- **+** *(plus sign)*: represents one or more of the preceding AS number
- **^** *(circumflex)*: represents a negation statement, used to exclude a set of AS numbers in conjunction with the square brackets
- ***[ ]*** *(square brackets)*: delimits a range or set of AS numbers
- ***“ ”*** *(quotation marks)*: used to enclose regular expressions, all expressions must be enclosed in “ ”
- ***( )*** *(parentheses)*: groups operators and explicitly conveys precedence
- *(space)*: represents the binary “and” operation, used to connect multiple expressions
- **|** *(vertical bar)*: represents the binary “or” operation
- **–** *(dash)*: represents the separation point for a range of AS numbers

## Using Regular Expressions

The following CLI command can be used to BGP RIB routes matching a particular regular expression.  In the next section, we will see how to use regular expressions in routing polices.

Syntax:
```
RS# bgp show regexp <regexp>
```

Example, show all routes with a single AS number in the AS path:
```
RS# bgp show regexp “.”
```

Example, show all routes that are advertised to us by AS 65050:
```
RS# bgp show regexp “65050 .*”
```

Example, show all routes with an origin of AS 65100 (the origin AS is always the last AS number in the AS path):
```
RS# bgp show regexp “.* 65100”
```

Example, show all routes with any AS path:
```
RS# bgp show regexp “.*”
```

Example, show all routes with at least two AS numbers:
```
RS# bgp show regexp “. .+”
```

Example, show all routes with no less than one and no more than two AS numbers:
```
RS# bgp show regexp “.?”
```

Example, show all locally generated routes (empty AS path):
```
RS# bgp show regexp “”
```

Example, show all routes that traversed AS 65499:
```
RS# bgp show regexp “.* 65499 .*”
```

Example, show all routes with only 65499 or 65000 in the AS path:
```
RS# bgp show regexp “65499|65000”
```

Example, show all routes that have at least one AS number and do not end with the AS path “65000 65001”:
```
RS# bgp show regexp “.* [^65000 65001]”
```

Example, show all routes that only contain public AS numbers (exclude routes with private AS numbers):
```
RS# bgp show regexp “.* [^64512-65535]* .*”
```

Example, show all routes that include the AS path “65000 65100” and don’t contain AS 65499 as the next AS number:
```
RS# bgp show regexp “65000 65100 [^65499] .*”
```

Example, show all routes with AS 65499 excluding ones that contain the AS path “65499 65020”:
```
RS# bgp show regexp “.* 65499 [^65020] .*”
```

---

# ***Routing Policies***

## Route Maps

Route maps enable route classification based on BGP path attributes.  They are used to define routing policies for importing routes, route selection and exporting routes for BGP.  Route maps are essential for traffic engineering and prefix management and gives you powerful fine-grained control over prefixes, whereas the global routing policies work at a protocol level.  The policies can be applied to a peer group or a peer host, and can be applied as import (received from a peer) or export (advertising to a peer) policies.  A route map has three steps that are used to evaluate prefixes.

1. Matching path attributes
2. Taking an action based on the match: permit or deny
3. Setting path attributes (optional)

Prefixes are evaluated by route maps until a matching statement is found, the permit or deny action is taken, then the set statement is executed.  Evaluation then stops unless combined with another policy.  A prefix that does not match any sequence is not used (implicit deny).  Permit or deny sequence are used to connect multiple statements which can contain multiple matches and sets in each sequence.  For added flexibility, route maps can be logically ANDed with other route maps by nesting them, or by applying multiple route maps to a peer or group.

Each time a policy change is made, the prefixes are re-evaluated as needed (don’t have to soft clear the peer like on a Cisco router).

### General Route Map Syntax

The route map must be created first, before it can be applied to any peers.  Several match and set options are supported, which are discussed in the following section.

Syntax:
```
RS(config)# route-map <name> permit|deny <sequence> [match options] [set options]
```

Example, deny the prefix 10.0.0.0/8, and permit all other prefixes:
```
RS(config)# route-map DENY_ROUTES deny 10 match-prefix network 10.0.0.0/8
RS(config)# route-map DENY_ROUTES permit 20 match-prefix network all
```

A route map can be applied as an import policy using the following syntax.

Syntax:
```
RS(config)# bgp set peer-host <IP address> route-map-in <name> in-sequence <sequence>
RS(config)# bgp set peer-group <name> route-map-in <name> in-sequence <sequence>
```

Example:
```
RS(config)# bgp set peer-host 10.0.0.2 route-map-in DENY_ROUTES in-sequence 10
RS(config)# bgp set peer-group VVNet route-map-in DENY_ROUTES in-sequence 10
```

Or, a route map can be applied as an export policy.

Syntax:
```
RS(config)# bgp set peer-host <IP address> route-map-out <name> out-sequence <sequence>
RS(config)# bgp set peer-group <name> route-map-out <name> out-sequence <sequence>
```

Example:
```
RS(config)# bgp set peer-host 10.0.0.2 route-map-out DENY_ROUTES out-sequence 10
RS(config)# bgp set peer-group VVNet route-map-out DENY_ROUTES out-sequence 10
```

**General Syntax Hints**

- Sequence numbers can be any range of numbers, but choose numbers wisely so that you can insert policies in the middle later on for example
  - 10, 20, 30, 40, …
  - 100, 200, 300, 400, …
  - Don’t use 1, 2, 3, 4, … because you would have to redo the entire policy if you need to insert a sequence between 2 and 3

- Policy names can be any combination of characters, but choose logical names that denote context so that operators can quickly identify it during configuration or troubleshooting
  - Include the peer/customer name in the policy’s name and denote if it is an import or export policy
  - For VVNet transit, name the policy “`VVNET_TRANSIT_IN`”
  - For a customer named Bob’s Bait Shop, name the policy “`BOBSBAITSHOP_OUT`”

**Route Map Match Options**

The following options can be used to match a prefix in a route map.

- `match-aspath-regular-exp`: matches either an aspath-list (discussed below) or a regular expression
- `match-community-list`: matches the communities defined in a community list (discussed below)
- `match-community-list-exact`: same as above, but perform an exact match
- `match-local-preference`: matches local preference
- `match-metric`: matches MED
- `match-next-hop-list`: matches a next hop IP address or a list of next hops
- `match-origin`: matches BGP origin
- `match-prefix`: matches an IP filter (discussed below) or a single prefix
- `match-prefix-list`: matches an IP prefix list (discussed below)
- `match-route-map`: matches a nested route map
- `match-route-type`: matches the routing protocol type (RIP, OSPF, IS-IS, BGP, etc)

**Route Map Set Options**

The following path attribute and parameters can be set after a successful match.

- `set-as-path-prepend`: prepends the specified AS path (always prepend your own AS, it is not nice to prepend someone else's AS)
- `set-community-list`: sets or adds the communities to either the communities in a community list (discussed later) or the communities specified in quotes, or removes the communities
- `set-dscp`: sets IP DiffServ DSCP value on matching flows
- `set-internal-metric`: sets the MED to the IGP metric
- `set-local-preference`: sets the local preference
- `set-metric`: sets the MED, +/- can be used for MED arithmetic
- `set-next-hop`: sets BGP next hop address
- `set-origin`: sets BGP origin
- `set-preference`: sets routing protocol preference
- `set-traffic-index`: sets a traffic bucket number for BGP accounting

Route maps can be displayed with the following CLI command.

Syntax:
```
RS# route-map show <name>|all
```

Example:
```
RS# route-map show all

route-map DENY_ROUTES, deny, sequence 10
Match clauses
prefix 10.0.0.0/8

Set clauses
route-map DENY_ROUTES, permit, sequence 20
Match clauses
prefix 0.0.0.0/0

Set clauses
```

Several additional policy building blocks are discussed in the next sections. These can be used in combination with route maps to define a list attributes that can be referenced in a route map. The building blocks provide separation and abstraction, and the ability to reuse smaller policy blocks to build complex routing policies.

## Prefix Lists

Prefix lists are used in combination with route maps to define a list of prefixes that can be referenced in a route map. You can use IP prefixes directly in a route map, but prefix lists provide abstraction and modularity. Two types of prefix lists are available: simple and extended.

### Simple Prefix Lists

This policy should be used for simple best matching when no granular permit or deny sequence is needed.

Syntax:
```
RS(config)# ip-router policy create filter <name> network <network/mask>
RS(config)# ip-router policy add filter <name> network <network/mask>
```

Example, create a list with network 10.0.0.0/8:

```
RS(config)# ip-router policy create filter PERMIT_TEN_NET network 10.0.0.0/8
```

As with global routing policies, the default options are *exact* and *refines*. The switches *exact*, *refines*, *between*, and *restrict* can be used for additional control. Next, the prefix list must be reference in a route map using the following match option.

Example:

```
RS(config)# route-map TRANSIT_FILTER permit 10 match-prefix PERMIT_TEN_NET
```

Prefix lists can be displayed with the following CLI command.

Syntax:

```
    RS# ip-router show filter [name <name>]
```

Example:

```
    RS# ip-router show filter
    Filter [PERMIT_TEN_NET]:

    Prefix             Range        Flags
    10.0.0.0/8
```

**Extended Prefix Lists**

Extended prefix lists use a permit and deny sequence of prefix statements for added control in matching logic needed to do first matching. Evaluation works the same as for route map sequences, including an implicit deny at the end.

Syntax:

```
RS(config)# prefix-list <name> permit|deny <sequence> <network/mask> [le|ge <mask>]
```

The default behavior is an exact match, use the *le* and *ge* options to match subnets.

Example, deny 10.0.0.0/8, permit 172.16.0.0/12 and 192.168.0.0/16:

```
RS(config)# prefix-list PERMIT_ROUTES deny 10 10.0.0.0/8
RS(config)# prefix-list PERMIT_ROUTES permit 20 172.16.0.0/12
RS(config)# prefix-list PERMIT_ROUTES permit 30 192.168.0.0/16
```

Next, the prefix list must be referenced in a route map using the following match option.

Example:

```
RS(config)# route-map TRANSIT_FILTER permit 20 match-prefix-list PERMIT_ROUTES
```

Extended prefix lists can be displayed with the following CLI command.

Syntax:

```
    RS# prefix-list show [<name>] [sequence]
```

Example:

```
    RS# prefix-list show
    prefix-list PERMIT_ROUTES: 3 entries
    seq 10 deny 10.0.0.0/8
    seq 20 permit 172.16.0.0/12
    seq 30 permit 192.168.0.0/16
```

### Summary

Prefix lists summary for matching options (setting attributes does not apply in this case).

- Network/mask in route map: `match-prefix network` …
- Simple (best match): `ip-router policy create filter` …
- Extended (first match): `prefix-list` …

## Community Lists

Community lists are used in combination with route maps to define a list of communities that can be referenced in a route map.  You can use communities directly in a route map, but community lists provide abstraction and modularity. Well-known communities are supported by name (`no-export`, `no-advertise`, and `no-export-subconfed`).

Two types of community lists are available: simple and extended.  Each community list supports RFC 1991 communities and extended communities.  Both types of community can be mixed together in each type of list.

### Simple Community Lists

This policy should be used for simple best matching when no granular permit or deny sequence is needed.  It is created as follows.

Syntax:
```
RS(config)# ip-router policy create community-list <name> "<communities>"
RS(config)# ip-router policy add community-list <name> "<communities>"
```

Example, create a list list with communities "65000:500 no-export":
```
RS(config)# ip-router policy create community-list DENY_EXTERNAL "65000:500 no-export"
```

Then, a route map can be created that uses the community list.

Example:
```
RS(config)# route-map TRANSIT_FILTER permit 10 match-community-list DENY_EXTERNAL
```

### Extended Community Lists

Extended community lists use a permit and deny sequence of communities for added control in matching logic. Evaluation works the same as for route map sequences, including an implicit deny at the end.

Syntax:
```
RS(config)# community-list <name> permit|deny <sequence> "<communities>"
```

Example, deny communities "65000:100", permit "65000:600 65000:700", and permit "65000:1000":
```
RS(config)# community-list PERMIT_EXTERNAL deny 10 "65000:100"
RS(config)# community-list PERMIT_EXTERNAL permit 20 "65000:600 65000:700"
RS(config)# community-list PERMIT_EXTERNAL permit 30 "65000:1000"
```

Now the community list may be used in a route map.

Example:
```
RS(config)# route-map TRANSIT_FILTER permit 10 match-community-list PERMIT_EXTERNAL
```

**Summary**

Both type of community lists can be displayed with the following CLI command.

Syntax:
```
    RS# bgp show community-list [<name>]
```

Example:
```
    RS# bgp show community-list
    Name           Action Sequence Count Community List
    ============== ====== ======== ===== ==============
    PERMIT_EXTE... deny   10       1     65000:100
                   permit 20       2     65000:600 65000:700
                   permit 30       1     65000:1000
    DENY_EXTERNAL  permit 0        2     65000:500 no-export
```

Matching behavior is controlled by configuring `match-community-list` (contains the communities) or `match-community-list-exact` (matches communities exactly) in the route map. Configuring `set-community-list` in the route map sets communities to the only specified list, and the `additive` switch can be used to add communities.

Community lists summary for matching options.

- Simple: `ip-router policy create community-list` …
- Extended: `community-list` …

Community lists summary for setting options.

- Community string in route map: `set-community-list` …
- Simple: `ip-router policy create community-list` …

## AS Path Lists

AS path lists are used in combination with route maps to define a list of AS path regular expressions that can be referenced in a route map. You can use regular expressions directly in a route map, but AS path lists provide abstraction and modularity. Two types of AS path lists are available: simple and extended.

### Simple AS Path Lists

This policy should be used for simple matching when no granular permit or deny sequence is needed. It is created as follows.

Syntax:
```
    RS(config)# ip-router policy create aspath-regular-expression <name> <regexp>
    RS(config)# ip-router policy add aspath-regular-expression <name> <regexp>
```

Example, create a regexp containing ".* 65015 .*":
```
    RS(config)# ip-router policy create aspath-regular-expression ANY_65015 ".* 65015 .*"
```

Once the AS path list is created, it can be referenced in a route map.

Example:
```
    RS(config)# route-map TRANSIT_FILTER permit 10 match-aspath-regular-expression ANY_65015
```

### Extended AS Path Lists

Extended AS path lists use a permit and deny sequence of regular expression statements for added control in matching logic needed to do first matching. Evaluation works the same as for route map sequences, including an implicit deny at the end.

Syntax:
```
    RS(config)# aspath-list <name> permit|deny <sequence> <regexp>
```

Example, deny ".* 65000", and permit ".* 65000 .*":
```
    RS(config)# aspath-list PERMIT_65000 deny 10 ".* 65000"
    RS(config)# aspath-list PERMIT_65000 permit 20 ".* 65000 .*"
```

Once the AS path list is created, it can be referenced in a route map.

Example:
```
    RS(config)# route-map TRANSIT_FILTER permit 10 match-aspath-regular-expression PERMIT_65000
```

### Summary

Both type of AS path lists (simple and extended) can be shown with the following CLI command.

Syntax:
```
    RS# bgp show aspath-regular-expression [<name>]
```

Example:
```
    RS# bgp show aspath-regular-expression
    Name           Action Sequence Regular Expression
    ============== ====== ======== ==================
    PERMIT_65000   deny   10       (.* 65000)
                   permit 20       (.* 65000 .*)
    ANY_65015      permit 0        (.* 65015 .*)
```

AS path lists summary for matching options (setting the AS path does not make sense).

- Regexp in route map: `match-aspath-regular-expression` …
- Simple: `ip-router policy create aspath-regular-expression` …
- Extended: `aspath-list` …

# *BGP Design Principles*

Building on the basic BGP configuration tasks, in this section several Best Current Practices are presented that focus on good real-world design principles. Topics such as good peering practices, filtering and route flap damping are discussed and should be used to implement configurations that provide network stability.

## Peering BCPs

BGP is very complex! Configuring BGP CLI commands is only one small step in implementing BGP, network operators must understand the protocol in order to maintain and troubleshoot networks. In addition, operators must understand the route selection process and how path attributes are used in order to build routing policies. The first step in implementing BGP is to determine if BGP is even needed. When one or more default or static routes will work, there is no need to run BGP. This will make the configuration and maintenance much simpler, and much less complex. General guidelines are as follows.

- Provide Internet transit: maybe, default and static routes may work too
- Need Internet routes for redundancy or load sharing: yes
- Need path attributes for routing policies: yes
- Multi-homed with different transit providers: maybe, default and static routes may work too

- Single-homed with one transit connection: no

There is significant motivation for implementing good peering practices. When configuring IGPs, mistakes that you make generally only affect your network and customers. However, since BGP is an inter-domain protocol, mistakes that you make can cause severe problems on the Internet. Don't become the next AS 7007! AS 7007 deaggregated tens of thousands of prefixes and leaked them to the Internet on April 25, 1997. A combination of unexplained software behavior and bad practices allowed this to cause havoc on the Internet for the better part of a day. Filtering and basic good peering practices would have prevented this problem. See http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html for more details about what happened

## IBGP and EBGP Peering Rule of Thumb

IBGP should always be configured between router loopback addresses. The loopback address is the most stable interface, it never goes down and is not susceptible to physical failures. Also, there is often more than one internal path to the loopback address in the network. So, this configuration will keep the IBGP session stable by using IGP to route around link failures without tearing down the IBGP session.

EBGP should be configured between interface addresses in most cases, since there is usually only one link between external routers. If a link fails, the peering session will be terminated immediately, without waiting the for *Hold* timer to expire. Another added benefit is that this configuration will not rely on the IGP for reaching the peer address, so instabilities in the IGP will not affect EBGP peering or cause flapping..

Several options are available that can be used to implement good peering practices and are discussed in this section. Options can be applied to a peer-host or peer-group, and multiple options can be combined on one line. The general syntax is as follows.

Syntax:

```
RS(config)# bgp set peer-host <IP address> <options>
RS(config)# bgp set peer-group <name> <options>
```

## Peer Configuration BCPs: `log-up-down`

Setting this option will enable the router to log neighbor state changes to the console and syslog. It is also a good idea to configure the NMS to receive SNMP traps, which are sent when a neighbor's state changes.

Example:

```
RS(config)# bgp set peer-group VVNet log-up-down
```

Example log messages are shown below.

```
%BGP-W-PEERUPDOWN, Peer 10.0.0.2 (External AS 65015) -- old state
Established, event Closed, new state Idle
%BGP-W-PEERUPDOWN, Peer 10.0.0.2 (External AS 65015) -- old state
OpenConfirm, event RecvKeepAlive, new state Established
```

## Peer Configuration BCPs: `max-prefixes`

This option limit the maximum number of prefixes that can be received from a peer. It protects you against someone else's instabilities that cause an extraordinary amount of prefixes to be advertised to you. For example, a customer usually advertises five prefixes but then turns up a transit peer and accidentally advertises 100,000 prefixes to your router.

Example, limit to 10 prefixes and log a warning message at 90% of the limit:

```
RS(config)# bgp set peer-host 10.0.0.2 max-prefixes 10 max-
prefixes-threshold 90
```

Example log messages are shown below.

```
%BGP-W-MAXPREF_THRESH, Count of BGP routes received from 10.0.0.2
(External AS 65015) reached 90 percent of the configured value.
%BGP-W-MAXPREF_EXCEED, Count of BGP routes received from 10.0.0.2
(External AS 65015) exceeds the configured value, Excess prefixes
dropped.
```

By default, excess prefixes over the limit are dropped. If the excess prefixes should be accepted, then the "`max-prefixes-warn-only`" switch can be added to only log a warning message without dropping prefixes. Alternatively, the "`max-prefixes-reset-sessi`" switch can be added to reset the session if the limit is reached. This is not recommended because it will probably cause repeated flapping.

## Peer Configuration BCPs: `password`

MD5 authentication can be used between BGP peers to protect the peering sessions. Each TCP segment is authenticated using an MD5 hash, and a password which is only known to each peer. Authentication can be used to prevent TCP spoofing and injection of bogus routing information. Peering will not come up if the password is wrong or missing.

Example:
```
RS(config)# bgp set peer-group IBGP password "secret password
phrase"
```

## Peer Configuration BCPs: `description`

This command allows an operator to add a text description to the peer that contains important information. It allows you to keep relevant information close to where you need it for when you need it, for example during troubleshooting.

Example:
```
RS(config)# bgp set peer-host 10.0.0.2 description "VVNet transit
provider, NOC 888-555-1234"
```

# Filtering BCPs

It is a good idea to configure filters on all EBGP peers to protect yourself from accepting bogus routing information. Healthy paranoia is your friend in these peering configurations, as you have very little control over other networks and thus are subject to receiving whatever routing information might be sent. AS path filters can be used, but in most cases this is not granular enough because a valid AS still can inject invalid routing information that AS path filters will not block. Thus, filtering by IP prefix is highly recommended.

Always apply filtering policies in both directions, this way you have redundant filters in case one is wrong or is removed. You apply import filters on your router, they apply export filters on their router. You apply export filters on your router, they apply import filters on their router.

## Bogon Filters

Basic filters should deny IP prefixes for routes that should never be advertised on the Internet. These are sometimes called "Bogon" filters. Each should be applied in import and export policies to each EBGP peer. Networks that should be filtered includes the following ones, which are nicely summarized in draft-manning-dsua-08.txt.

- RFC1918 addresses (reserved)
- 127.0.0.0/8 (loopback)
- Class D and E addresses
- Various other reserved addresses
- Your own IP address aggregates (import only)

Example bogon filter:
```
ip-router policy create filter bogons network 0.0.0.0/8
  ip-router policy add filter bogons network 10.0.0.0/8
  ip-router policy add filter bogons network 127.0.0.0/8
  ip-router policy add filter bogons network 192.168.0.0/16
  ip-router policy add filter bogons network 172.16.0.0/12
  ip-router policy add filter bogons network 169.254.0.0/16
  ip-router policy add filter bogons network 192.0.2.0/24
  ip-router policy add filter bogons network 192.88.99.0/24
  ip-router policy add filter bogons network 224.0.0.0/3
```

**Additional Filtering Recommendations**

Certain /8 address space is not yet allocated, and should not be advertised on the Internet. If you are really paranoid about accepting bogus routing information you can also create filters for unallocated addresses. However, these must be maintained vigilantly in case IANA allocates a new /8 to a regional registry, which happens from time to time. The list of allocated address space is at http://www.iana.org/assignments/ipv4-address-space/.

Filtering based on prefix length is a also good idea. This will protect you against IGP leaking, deaggregation, routing table pollution and other accidents. A /24 is generally the longest mask that should be accepted. Some providers only accept much shorter masks, for example a /19 or /20.

Example, don't accept any prefixes with a mask longer than /24:

```
RS(config)# bgp set peer-host 10.0.0.2 max-prefix-len 24
```

## Route Redistribution and BGP Next Hop BCPs

Several general rules for route redistribution should be followed, with very few exceptions. Following these steps will create the most stable and scalable network

- Do not redistribute BGP into an IGP, IGPs will not scale and doesn't carry path attributes
- Do not redistribute an IGP into BGP, keep internal routing information separate and only advertise aggregates

- Do not carry customer routes in your IGP, keep external routing information separate
- Only redistribute aggregates into BGP

BGP next hop address must be reachable in order for the router to use a BGP route.  This reachability is usually provided by the IGP.  Thus, changes in the IGP can influence forwarding to a BGP next hop address over the backbone, such as blackholing or forwarding problems.  The best way to avoid this problem is to keep your IGP lean and stable.  How to do this is somewhat of a matter of opinion, but there are three approaches you can take to ensure next hop reachability.

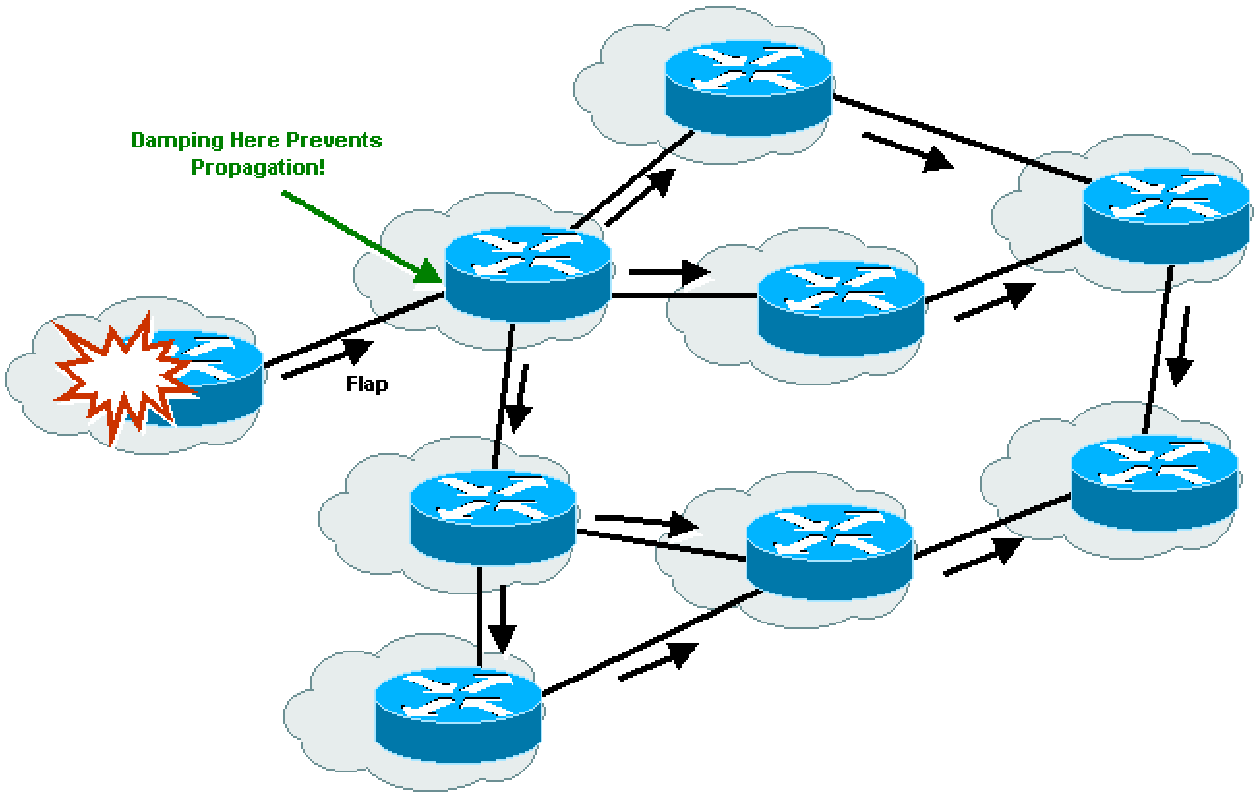
- Enable the IGP on external links, using the "`passive`" setting on each interface; this way no IGP are adjacencies formed on external links and they appear as native IGP routes
- Redistribute connected interfaces into the IGP; in this case external links appear as externals to the IGP which can be undesirable
- Use the "`next-hop-self`" setting on IBGP sessions to hide external link addressing from the IGP; keeps the IGP small but hides external link addresses which can be useful for troubleshooting

---

# *BGP Design Principles*

## Route Flap Damping

Route flap damping is a reactive measure that can be taken to prevent route flaps from propagating across the Internet by selectively suppressing route advertisement. It is defined in RFC 2439. The general idea is that past history predicts future behavior, so misbehaving routes are penalized, while stable routes are advertised with minimal delay. Some route flapping is acceptable though, because routing information is allowed to change to reflect changes being made on the Internet. Thus, reasonable changes should not be penalized either. One flap within a few minutes does not necessarily indicate a problem, but five flaps within a few minutes probably indicates a problem.



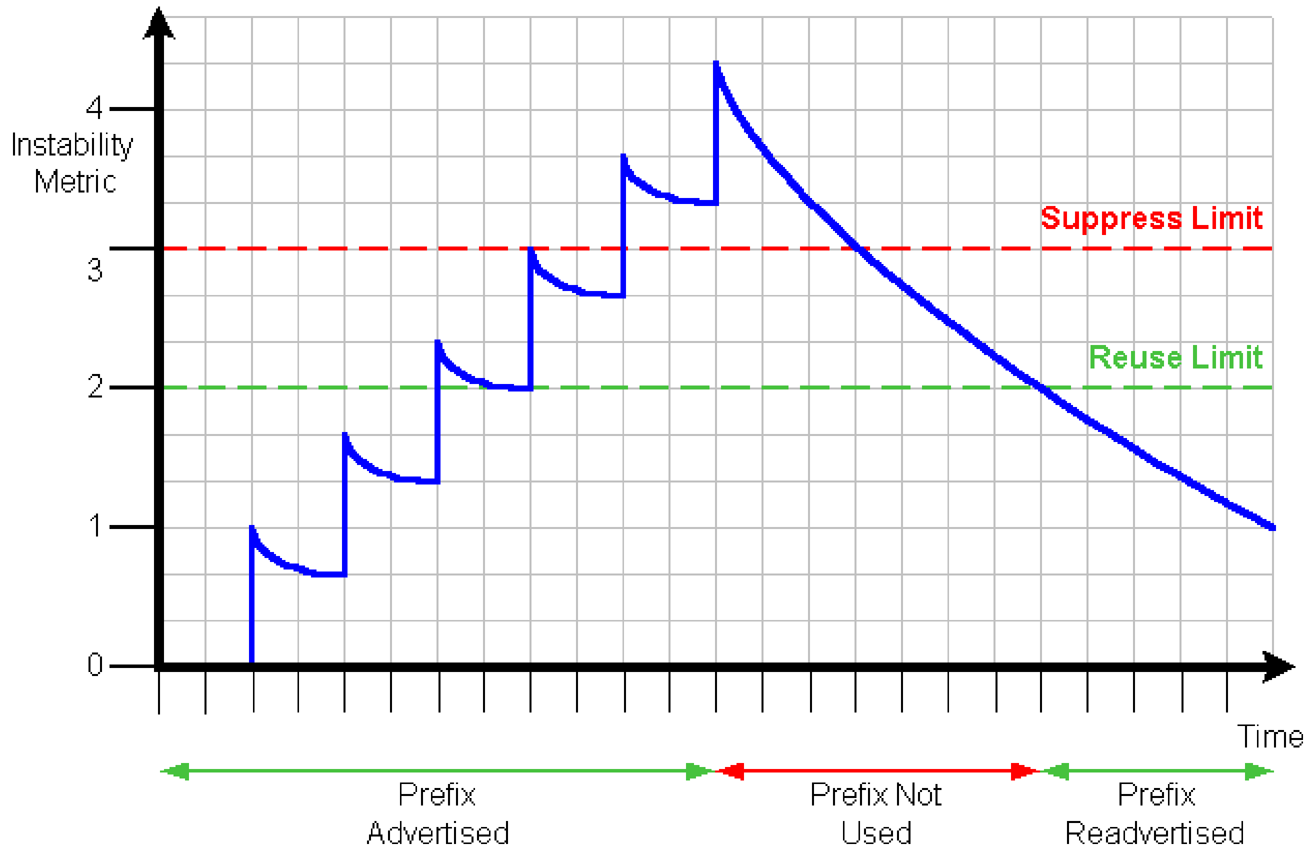
When damping is enabled, the flap history for a prefix is maintained in form of an instability metric that is a real number. When a prefix is withdrawn, the metric is incremented. The metric decays exponentially over time, faster when a prefix is reachable and slower when a prefix is unreachable. If the metric reaches a certain limit, the prefix is not advertised to other peers which stops the instability from propagating. Damping should only use for EBGP peering at network boundaries, so that the flapping can be suppressed as close as possible to the instability source.

**Definitions**

Several definitions are needed before the details of damping are explained.

- *Suppress Limit*: if the instability metric is above this value the prefix will be suppressed
- *Reuse Limit*: when the metric decays and reaches this value, a suppressed prefix will be reused
- *History* (h): the prefix has been withdrawn and is currently unreachable, a metric is maintained in case it starts flapping
- *Damped* (d): the prefix is currently reachable, and is used and advertised but has an instability metric from previous bad behavior that is below the suppress limit
- *Suppressed* (s): he prefix is currently reachable but has a metric that is above the suppress limit , it is not advertised to other peers and not installed in the FIB even though the prefix is reachable

It is important to note that a prefix is only suppressed when receiving an advertisement, not on a withdrawal. The advertisement must be received when the metric is greater than the suppress limit. This means that the metric can be greater than the suppress limit without causing the route to be suppressed. After all, if the prefix is behaving well then there is no need to suppress it.



A prefix starts out with a value of 0.0. Every time it flaps, its instability metric is incremented by 1.0. When the metric value is above 3.0, and the prefix is readvertised after a flap, it is suppressed. The instability metric then decays over time, for example every 5 minutes (300 seconds) the instability metric is halved while the prefix is reachable. Once the metric falls below 2 the route is advertised and used again, as shown in this example.

If you want to get creative, you can configure different damping policies by tuning the options for different situations. For example, the following different damping parameters could be used.

- Based on peer type: short damping period for customer routes, longer damping period for others; damping your customers takes them off of the Internet, so use a short time for customers that stops instability but removes the penalty as soon as stability is regained
- Based on prefix length: short damping period for ≤ /16, medium for ≤ /19 and longer for ≤ /24; damping a /16 is much more costly in terms of lost reachability then damping a /24, so used a shorter damping period for shorter masks

It is important to realized that damping is not the silver bullet against route flapping. Damping is a reactive measure taken against you by others because you are causing a problem or taken against your customers because they are causing a problem. Good aggregation and peering policies should be implemented to minimize instabilities first, so that damping is not even needed. An excellent document with recommendations based on operational experience by the RIPE Routing-WG is available at http://www.ripe.net/docs/ripe-210.html. It has some example policies that are worth reading.

## Damping Configuration

Damping can be enabled for a peer or for a group. It can also be enabled globally for all BGP peers, but this is almost certainly not what you want and is not recommended. Damping is configured with a route map, which can also match path attributes for more granularity in selecting which prefixes to select. First the damping parameters are defined, then the route map is applied as an import policy to a peer or group.

Syntax:
```
RS(config)# route-map <name> set dampenflap state enable <options>
```

Example:
```
RS(config)# route-map customer-damping set dampenflap state enable
```

Several options are configurable, but the defaults will do for most situations unless complex damping is needed.

- *max-flap*: upper limit of instability metric (default 16.0)
- *reuse-below*: value of the instability metric at which a suppressed route will be reused (default 2.0)
- *suppress-above*: value of the instability metric at which a prefix may be suppressed (default 3.0)
- *keep-history*: how long history is maintained for a prefix (default 1800 seconds/30 minutes)
- *reach-decay*: how long it takes for the metric to decay to half its value when a prefix is reachable (default 300 seconds/5 minutes)
- *unreach-decay*: how long it takes for the metric to decay to half its value when a prefix is unreachable (default 900 seconds/15 minutes)

## Route Flap Damping Example

In this example, damping has been configured for the VVNet peer using the default settings. The configuration is as follows.

```
route-map TRANSIT-DAMPING set dampenflap state enable
bgp create peer-group VVNet autonomous-system 65015
bgp add peer-host 10.0.0.2 group VVNet
bgp set peer-group VVNet route-map-in TRANSIT-DAMPING in-sequence 10
```

To show the prefixes that have a history, are damped or are suppressed use the following CLI command.

```
    RS# bgp show flap-statistics all
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                  s - suppressed, d - damped, h - history
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network             Next Hop        Metric Last-Flap Reuse    Path
        -------             --------        ------ --------- -----    ----
    h   4/8                 10.0.0.2        1.00   00:04:17
    *>d 172.16/12           10.0.0.2        0.96   00:00:17           (65030) 65015 i
    * s 192.0.2/24          10.0.0.2        4.92   00:00:03  00:07:00 (65030) 65015 i
```

To only show the prefixes that have a history and are currently unreachable, use this command.

```
    RS# bgp show flap-statistics history
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                  s - suppressed, d - damped, h - history
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network             Next Hop        Metric Last-Flap Reuse    Path
        -------             --------        ------ --------- -----    ----
    h   4/8                 10.0.0.2        1.00   00:05:41
```

Prefixes that are reachable, advertised and used can be displayed with the "`damped`" switch.

```
    RS# bgp show flap-statistics damped
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                  s - suppressed, d - damped, h - history
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network             Next Hop        Metric Last-Flap Reuse    Path
        -------             --------        ------ --------- -----    ----
    *>d 172.16/12           10.0.0.2        0.73   00:02:14           (65030) 65015 i
```

Suppressed prefixes can be shown with the following command.  In this example, the prefix will be reused in five minutes unless it flaps again.

```
    RS# bgp show flap-statistics suppressed
    Local router ID is 192.0.2.1
    Status codes: > - best, * - valid, i - internal, t - stale
                  s - suppressed, d - damped, h - history
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network             Next Hop        Metric Last-Flap Reuse    Path
        -------             --------        ------ --------- -----    ----
    * s 192.0.2/24          10.0.0.2        3.69   00:02:08  00:05:00 (65030) 65015 i
```

Damping information is also displayed in the RIB and for each prefix, as this example below shows.

```
RS# bgp show routes all
Local router ID is 192.0.2.1
Status codes: > - best, * - valid, i - internal, t - stale
              s - suppressed, d - damped
Origin codes: i - IGP, e - EGP, ? - incomplete

    Network            Next Hop        Metric LocPrf Path
    -------            --------        ------ ------ ----
*>d 172.16/12          10.0.0.2                  100 (65030) 65015 i
* s 192.0.2/24         10.0.0.2                  100 (65030) 65015 i

RS# bgp show routes 192.0.2/24
Local router ID is 192.0.2.1
BGP routing table entry for 192.0.2/24
Path: Not Best
Source: 10.0.0.2
Router Id : 10.0.0.2
Advertised to Tasks: None
Flap statistics: suppressed due to damping, metric 2.65 last flap 00:04:31 ago,
                reuse in 00:02:30
Local AS: 65030   Peer AS: 65015   Age: 4:31
NextHop: 10.0.0.2          MED:  -1    Local Preference: 100
AS Path: (65030) 65015 i
```

---

# Advanced Configuration Topics

Advanced topics such as the RS memory manager and route refresh are presented in this section.

## Memory Manager

The routing engine only has a finite amount of RAM available for use, as there is only a finite amount of RAM installed on a router.  This RAM is shared between the routing engine and all other functions on the router.  Thus, if the router starts to run out of memory, the memory manager starts to throw away duplicate RIB routes (never FIB routes) as a preventative measure.  The other alternative is to just run out of RAM and crash, so in this situation the memory manager tries to make the best decision and handle the situation gracefully.

Four memory threshold levels are used to determine what action to take, based the percentage of RAM available on the router.  No action is taken until the router starts to get low on memory

- Level 1: 70%
- Level 2: 73%
- Level 3: 76%
- Level 4: 80%

OSPF and IS-IS routing updates are always processed at each threshold level. BGP routes that are in the RIB may be dropped according to the following criteria.

- Level 1 and Level 2: a new BGP route is only added if it is the only BGP route to the destination; maximum of three routes are allowed to a destination, if there are more than three routes to a given destination, a new route replaces an existing one
- Level 3: maximum of two routes allowed to a destination, if there are more than two routes to a given destination, a new route replaces an existing route
- Level 4: no new BGP routes are added

Thresholds can be changed with the following command.

Syntax:
`RS#(config)` **`ip-router global set memory-threshold level-1`** ***<percentage>*** **`level-2`** ***<percentage>*** **`level-3`** ***<percentage>*** **`level-4`** ***<percentage>***

Example, set the levels to 75, 78, 82 and 85:
`RS#(config)` **`ip-router global set memory-threshold level-1 75 level-2 78 level-3 82 level-4 85`**

The default values are probably right for most networks. You can tune them a little to make more efficient use of the available RAM, if there are no other RAM-hungry processes (RMON, for example) on the router. It is not recommended to set thresholds above 85%, because this could impact other functions. The memory manager can be disabled, but this is <u>not</u> recommended and <u>will</u> cause the router to crash if it runs out of RAM.

Warning messages are logged to the console and syslog when thresholds are reached.

```
    2002-01-23 12:26:43 %ROSRD-W-LOWMEMTHRHIT, Threshold 1
    reached.
    2002-01-23 12:26:45 %SYS-W-HEAP75FULL, heap is at 75 percent
    of maximum usage
    2002-01-23 12:26:48 %ROSRD-W-LOWMEMTHRHIT, Threshold 2
    reached.
```

```
2002-01-23 12:26:50 %ROSRD-W-LOWMEMTHRHIT, Threshold 3
reached.
2002-01-23 12:26:57 %ROSRD-W-LOWMEMTHRHIT, Threshold 4
reached.
```

If the router decides to drop BGP routes, the following messages are logged.

```
2002-01-23 12:26:58 %ROSRD-W-LOWMEMDROPCT, 2048 routes dropped
due to low memory
2002-01-23 12:26:59 %ROSRD-W-LOWMEMDROPCT, 4096 routes dropped
due to low memory
2002-01-23 12:27:00 %ROSRD-W-LOWMEMDROPCT, 6144 routes dropped
due to low memory
2002-01-23 12:27:01 %ROSRD-W-LOWMEMDROPCT, 8192 routes dropped
due to low memory
2002-01-23 12:27:02 %ROSRD-W-LOWMEMDROPCT, 10240 routes
dropped due to low memory
2002-01-23 12:27:02 %ROSRD-W-LOWMEMDROPCT, 12288 routes
dropped due to low memory
2002-01-23 12:27:03 %ROSRD-W-LOWMEMDROPCT, 14336 routes
dropped due to low memory
```

The following CLI command can be used to monitor memory utilization, thresholds and drops.

```
RS# ip-router show summary drops

Summary of routes in RIB
------- -- ------ -- ---
Number of Unique routes : 81016
Number of routes        : 81016
Kernel routes           : 0
Direct routes           : 4
Static routes           : 1
RIP routes              : 0
OSPF routes             : 0
OSPF ASE routes         : 0
ISIS level 1 routes     : 0
ISIS level 2 routes     : 0
```

```
    BGP routes              : 81010
    Stale BGP routes        : 0
    Other Protocol routes   : 1
    Hidden routes           : 1
    Install LSP routes      : disabled
    Routes not added due to low memory          : 14336
    BGP Routes removed to accomodate new routes : 0
    BGP Routes removed due to low memory        : 0
    RIP Routes removed to accomodate new routes : 0
    RIP Routes removed due to low memory        : 0
    Low Memory Threshold 1                      : 70
    Low Memory Threshold 2                      : 73
    Low Memory Threshold 3                      : 76
    Low Memory Threshold 4                      : 80
    Current Threshold value                     : 4
    Malloc Percentage Used                      : 83
    Gated Free Memory                           : 1545792
    Gated Free Memory(Global counter)           : 1545792
    Gated Free Memory(Calculated)               : 1545792
    Gated Used Memory(Calculated)               : 31997408
    Gated Free memory checking                  : Enabled
```

## Route Refresh

Route refresh is an extension to BGP that is defined in RFC 2918. Using this feature, a BGP speaker can request a complete retransmission of the peer's NLRI without tearing down and reestablishing the BGP session, saving a route flap. It is used to facilitate routing policies changes, without storing an unmodified copy of the peer's routes on the local router to save RAM. The capability must be supported by both routers, and is negotiated using capabilities code 2 (Route Refresh Capability). When both routers in the peering session support this open, each router will respond to requests issued from the peer without operator intervention.

The capability should be used in combination with the "`delete-policy-rejects`" option in situations where large amounts of prefixes are filtered with an import policy. The "`delete-policy-rejects`" option will configure the router to delete prefixes that are rejected by a policy from the RIB to save RAM. Now instead of resetting the BGP session with a hard clear, you can request a route refresh if you make policy changes.

Syntax:

```
    RS# bgp clear peer-host <peer address> soft-inbound
```

Example:

```
    RS# bgp clear peer-host 10.0.0.2 soft-inbound
```

The following message will be logged when a route refresh is requested by a peer.

```
    2002-01-23 12:30:29 %BGP-I-RTREFRESHSENT, A route refresh
    request was sent to peer 10.0.0.2
```

Route refresh is enabled by default, and can be disabled by setting the "`no-route-refresh`" option on a peer or group. However, there probably no need to disable it, since capability negotiation will not permit a router to use a capability that both peers do not support.

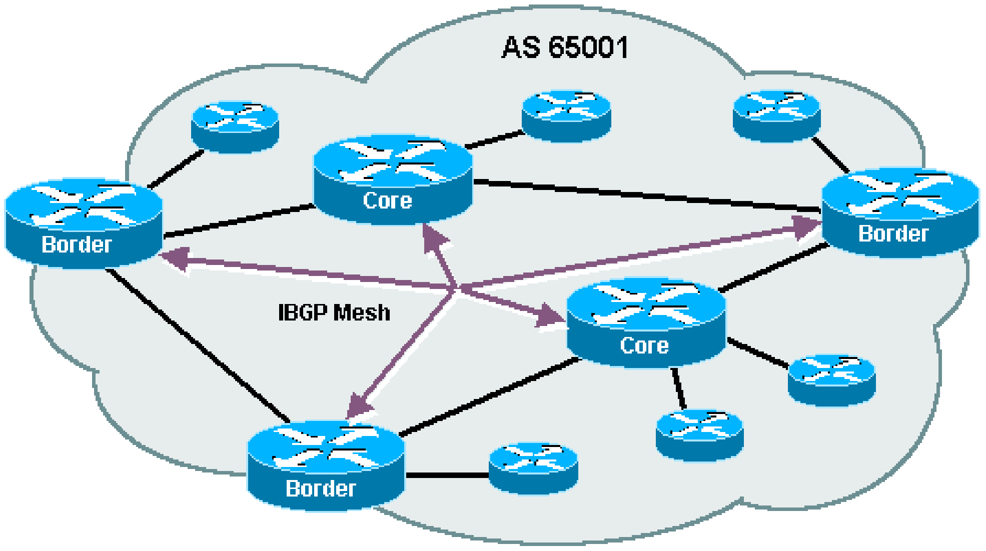
---

# *IBGP Scalability*

The IBGP full mesh problem is revisited in more detail, and two mechanisms to remove the full IBGP mesh requirement are introduced and explained in detail.

## IBGP Full Mesh Problem

Recall from previous sections that there is no loop detection in IBGP, and readvertising IBGP routes to an IBGP peer could cause a loop. This is because an IBGP speaker cannot prepend its own AS to the AS path, which would then trigger the loop detection. Border routers and core routers must be in the IBGP mesh to have a full view of transit routes so they can do best-exit routing and know about all next hops. In other words, each IBGP router must have a BGP session with all other IBGP routers (not necessarily a physical link, though). Unfortunately this full mesh requirement presents some scalability problems. Even a small ISP could have a number of routers that would make managing a full IBGP mesh very difficult.

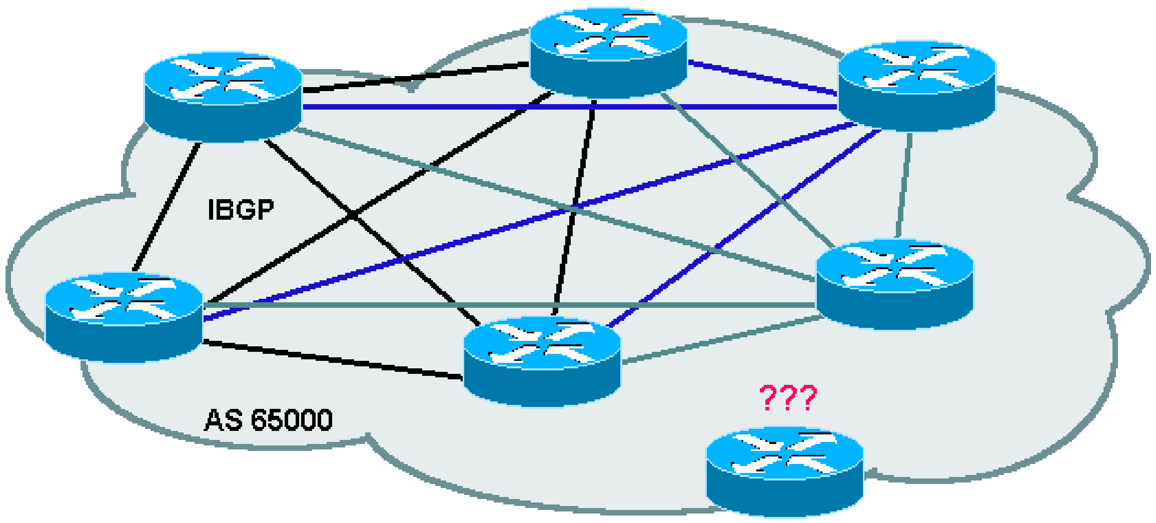


To give you an idea of how quickly a full mesh can become unmanageable, consider the following examples.

- 2 routers: (2)(2 - 1)/2 = 1 BGP session
- 3 routers: (3)(3 - 1)/2 = 3 BGP sessions
- 4 routers: (4)(4 - 1)/2 = 6 BGP sessions
- 10 routers: (10)(10 - 1)/2 = 45 BGP sessions
- 20 routers: (20)(20 - 1)/2 = 190 BGP sessions
- The number of BGP sessions needed can be expressed by this formula: $n$ routers: ($n$)($n$ - 1)/2

As seen in the example below, with only four routers it is already complicated before the fifth router is added. The bottom line is that full meshes do not scale well for several reasons.

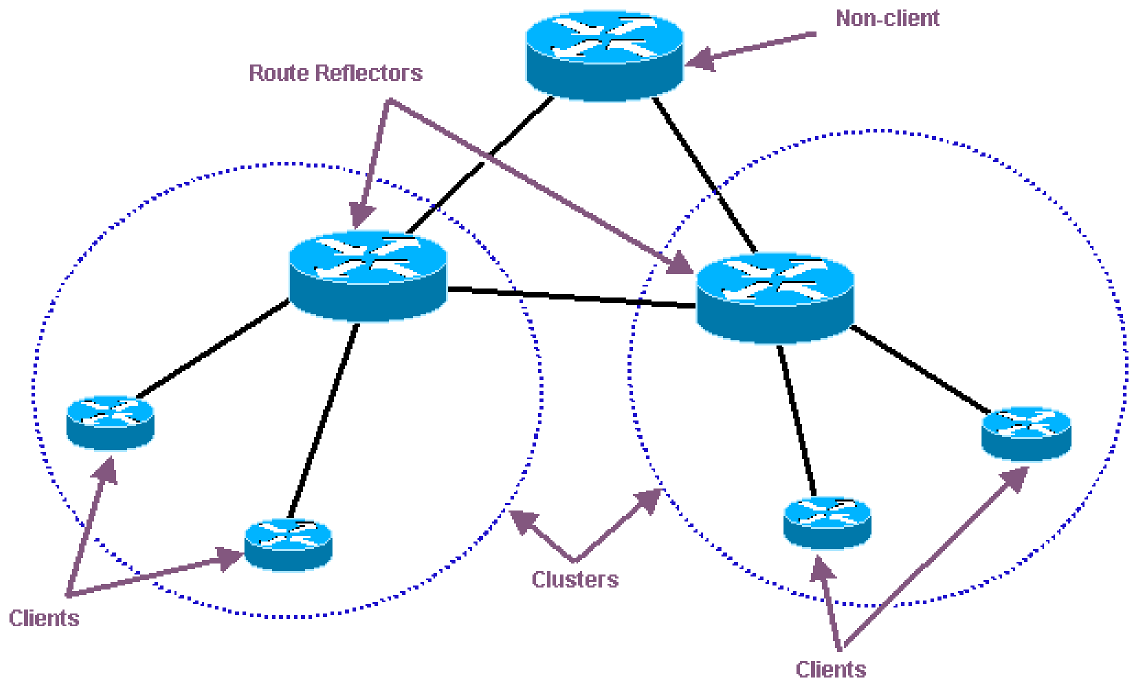
- Administration: must configure a new peer on every router in the mesh, this quickly becomes very time consuming as the number of routers grows
- Overhead: router must use CPU and RAM to setup and maintain resources for each peer, which can become quite costly as number of peers and routes increase



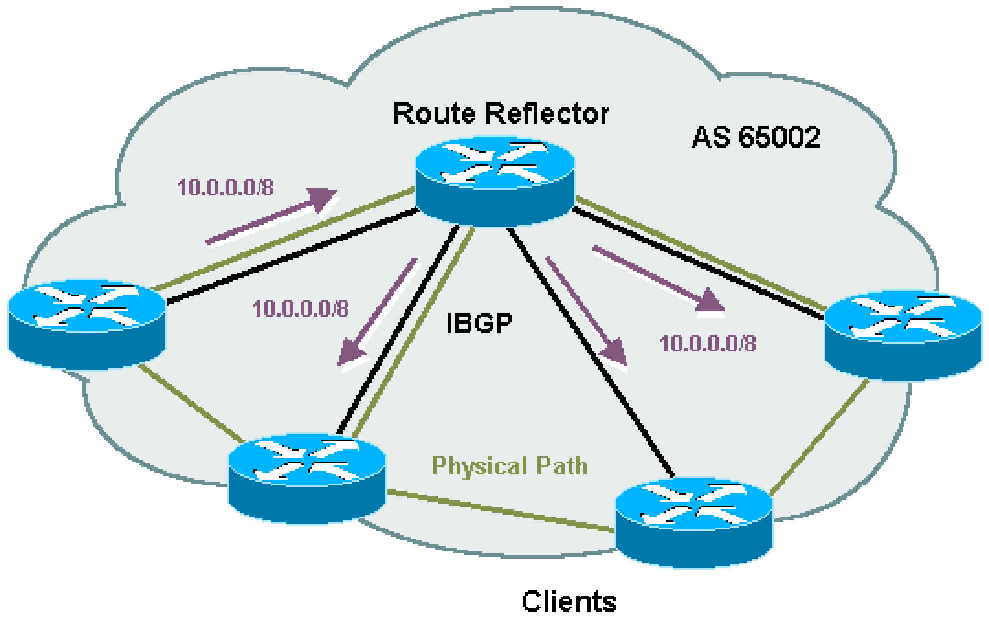
Unfortunately, not having a full IBGP mesh can cause big problems too, because some routers will not know about next hop addresses which can lead to blackholing and routing loops. Two mechanisms are supported to address IBGP full mesh problem: route reflection and confederations. Both are in use today and widely deployed, and each takes a different approach to IBGP scalability. Neither mechanism is inherently better, and some networks even use both together for maximum scalability.

## Route Reflection

Route reflection is defined in RFC 2796, and introduces a hierarchy approach to remove the full mesh requirement. Instead of peering with all other IBGP routers in the network, each IBGP router only peers with a router configured as a *route reflector*. Route reflectors are the only routers that are still fully meshed. Peers of the route reflector are called *clients*. The rules change for the route reflector, in this role it is allowed to redistribute IBGP routes to its IBGP reflector clients. Route reflector clients are not aware that they are peering with a route reflector, because their behavior does not change at all. Hierarchy is built through logical segmentation of groups of routers into areas called *clusters*.



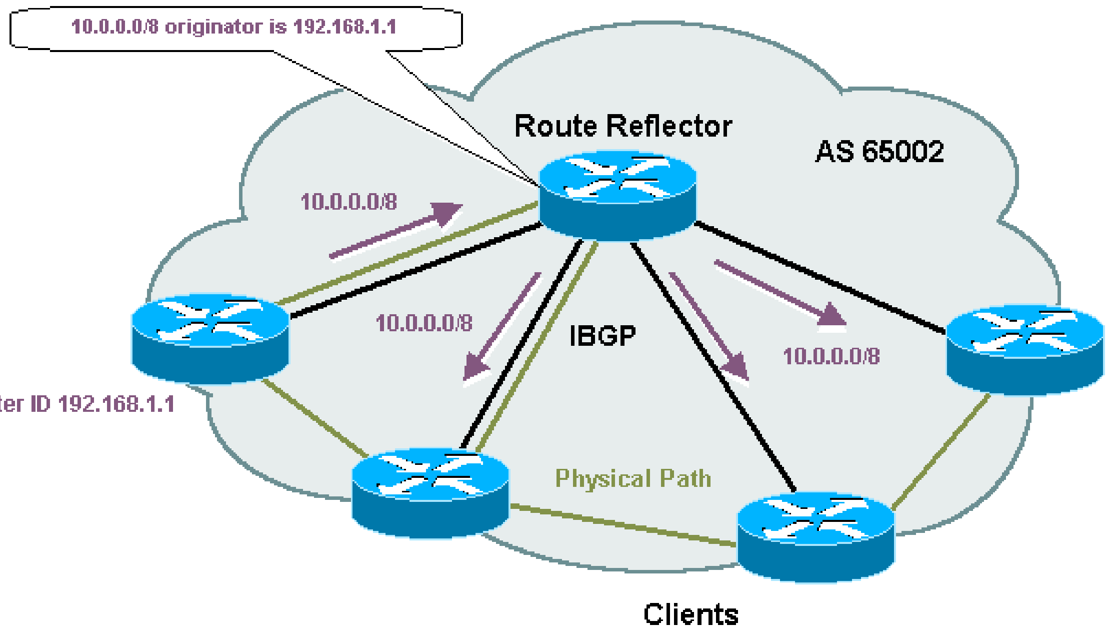
Each cluster has a route reflector that is used to distribute routes, and a unique *cluster ID* that identifies the cluster. Multiple route reflectors can be used within a cluster for redundancy. Only the best route is advertised between a route reflector and its clients, just as for IBGP and EBGP and the IGP is still needed to carry internal routes. Clever network designs sometimes overlay the IGP hierarchy with the route reflection hierarchy, for example clusters could each be a separate IGP area.



All path attributes are passed on to clients unmodified. This is especially important for the next hop address, as it must not be modified or else all traffic would go through the route reflector. Since a route reflector may now advertise IBGP routes to other IBGP peers, it is now possible for routing information to loop. Two new path attributes for route reflection are defined to prevent looping.

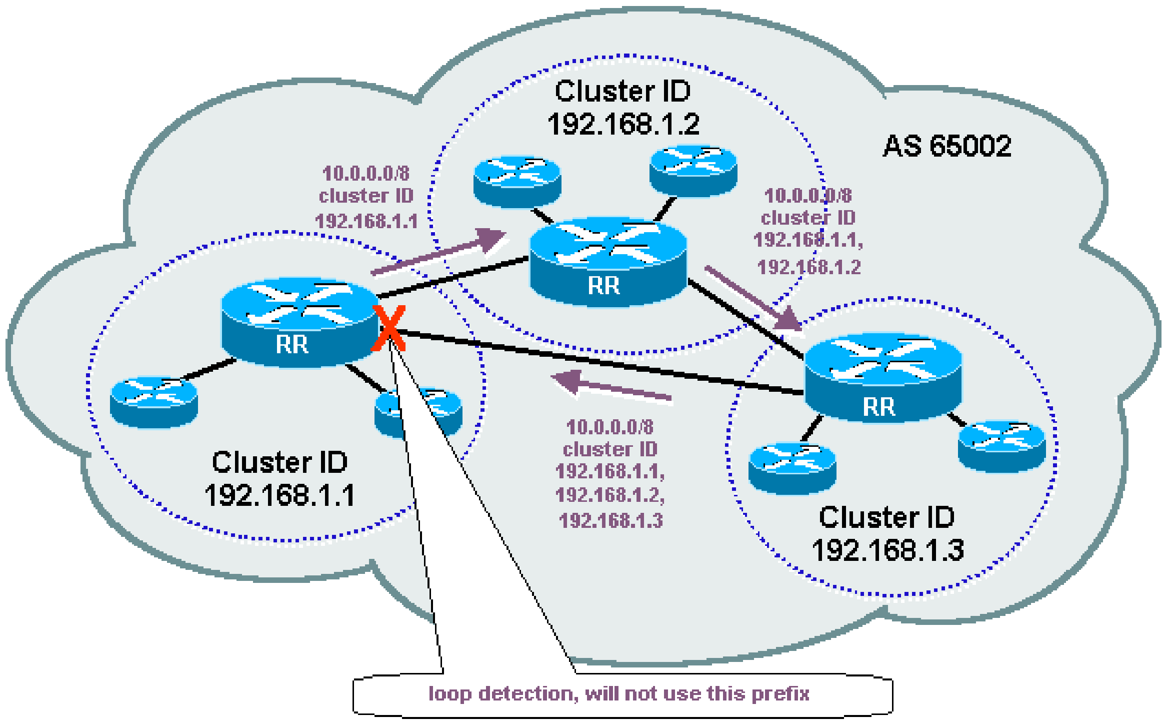
***ORIGINATOR-ID* (Type Code 9, Optional Non-transitive - RFC 2796)**

This path attribute defines the prefix's originating router. The route reflector sets the *ORIGINATOR-ID* to the router ID of the client that originates the prefix. Router reflectors will not advertise a prefix to a router if its router ID is the same as the *ORIGINATOR-ID* in order to avoid a routing information loop. Clients never set the *ORIGINATOR-ID*.



***CLUSTER-LIST* (Type Code 10, Optional Non-transitive - RFC 2796):**

The *CLUSTER-LIST* records the reflector clusters that a prefix has traversed. Route reflectors append the cluster ID to the *CLUSTER-LIST* when advertising the route outside of the cluster. If a route reflector detects it's own cluster ID in an advertisement from a peer, it does not accept the prefix, because this would cause a routing information loop. Clients never modify the *CLUSTER-LIST*.


Cluster ID 192.168.1.2
AS 65002
10.0.0.0/8 cluster ID 192.168.1.1
10.0.0.0/8 cluster ID 192.168.1.1, 192.168.1.2
RR
RR
RR
X
10.0.0.0/8 cluster ID 192.168.1.1, 192.168.1.2, 192.168.1.3
Cluster ID 192.168.1.1
Cluster ID 192.168.1.3
loop detection, will not use this prefix

## Route Reflection Configuration

Configuration is simple, as you only need to configure the route reflector. No configuration changes are needed on the clients. The following command enables a router to become a route reflector.

Syntax:
```
RS(config)# bgp set peer-group <name> reflector-client
```

Example:
```
RS(config)# bgp set peer-group IBGP reflector-client
```

Next, enable route redistribution for the route reflector.

Syntax:
```
RS(config)# ip-router policy redistribute from-proto bgp to-proto bgp source-as <AS> target-as
<AS>
```

Example:
```
RS(config)# ip-router policy redistribute from-proto bgp to-proto bgp source-as 65001 target-as
65001
```

The cluster ID will automatically be set to the router's router ID by default. If there is more than one route reflector in the cluster, then a cluster ID must be configured using the following command.

Syntax:
```
RS(config)# bgp set cluster-id <ID>
```

Example:
```
RS(config)# bgp set cluster-id 0.0.0.1
```

When route reflection is enabled for a group, the cluster ID is displayed in the output of the summary commands.

```
RS# bgp show globals
 Router ID                : 192.0.2.1
 Autonomous System        : 65001
 Cluster ID               : 0.0.0.1
 BGP default preference   : 170
 BGP default localpref    : 100
 BGP default metric       : -1
 Action on a bad aspath   : Reset session
 Comparison of MED        : Only for same AS

RS# bgp show summary
Local router ID is 192.0.2.1, Local AS number 65001
BGP Route Entries 3, Unique AS Paths 5
Unique Communities 0, Unique Extended Communities 0

Neighbor        V    AS MsgRcvd MsgSent      Up/Down Prefixes Rcvd/Sent
--------        -    -- ------- -------      ------- ------------------
[Group Id: IBGP]  Route Reflector enabled, cluster id is 0.0.0.1
192.0.2.2       4 65001      12      15    0d0h9m27s         1/3
192.0.2.8       4 65001      12      15    0d0h9m24s         1/3
192.0.2.7       4 65001      12      15    0d0h9m20s         1/3
BGP summary, 1 groups, 3 peers
```

The path attributes used for route reflection are also printed for each prefix, as shown below.

```
RS# bgp show routes 172.16.64/19
    Local router ID is 192.0.2.2
    BGP routing table entry for 172.16.64/19
    Path: Best
    Source: 192.0.2.1
    Router Id : 192.0.2.1
    Advertised to Tasks:
     3-KRT 5-BGP_Sync_IBGP
    Flap statistics: None
    Local AS: 65001   Peer AS: 65001   Age: 4:18
    NextHop: 192.0.2.7         MED:  -1    Local Preference: 100
    AS Path: (65001) ?
    Originator Id 192.0.2.7
    Cluster list: 192.0.2.1
```

### Route Reflection Summary

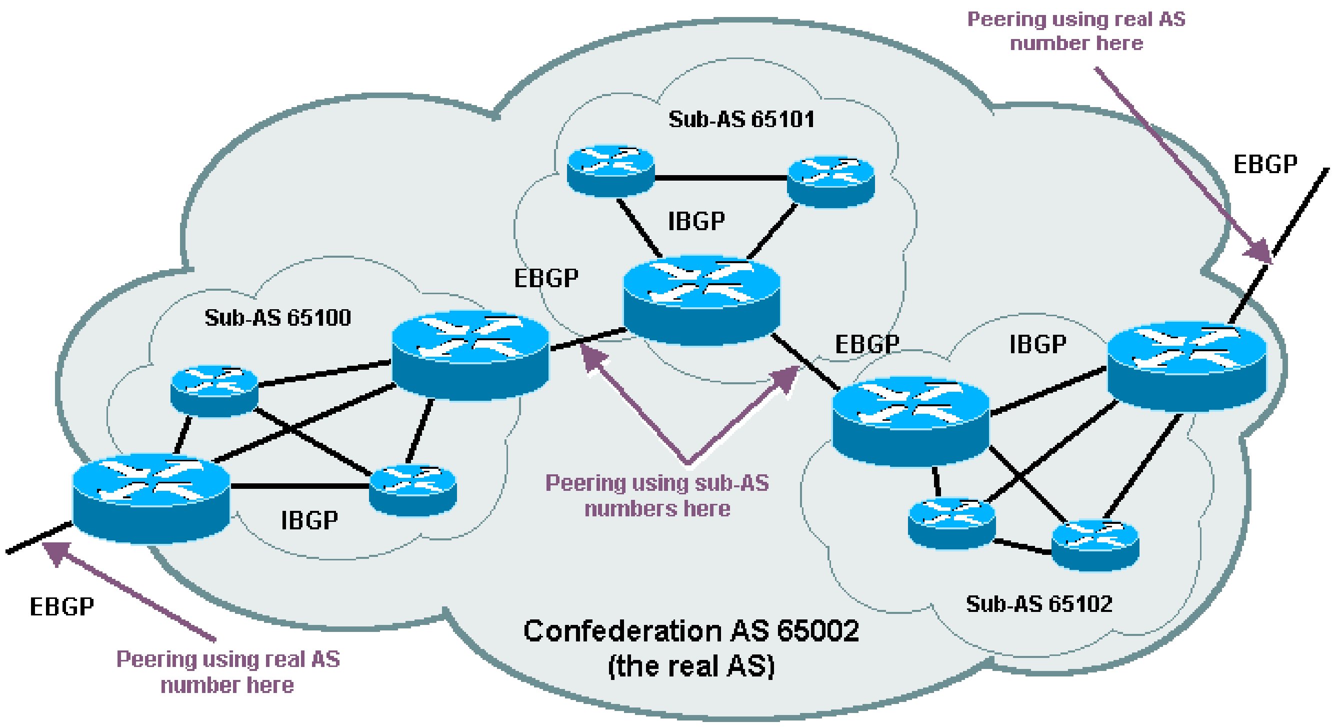
- Solves full IBGP mesh problem
- Clients and non-clients can coexist in the same cluster or network
- Hierarchy support
- Redundancy with multiple route reflectors
- Configuration is easy
- Migration from full IBGP mesh to route reflection is easy

---

# *IBGP Scalability*

## Confederations

Confederations are another way of scaling IBGP. Defined in RFC 3065, this feature introduces a divide-and-conquer approach to remove the full mesh requirement.
Using confederations an AS is split into multiple sub-ASs, but the network still appears as one AS to the outside world. Each sub-AS number is stripped from AS path at the confederation border. A full IBGP mesh is only required within each sub-AS, which is usually a manageable number of routers. In very large networks, you can even configure route reflection within a sub-AS. Typically private ASs are assigned for each sub-AS number.


Peering using real AS number here
Sub-AS 65101
EBGP
IBGP
EBGP
Sub-AS 65100
EBGP
IBGP
IBGP
Peering using sub-AS numbers here
EBGP
Sub-AS 65102
Confederation AS 65002 (the real AS)
Peering using real AS number here

A modified EBGP peering is used between sub-ASs that some changes to the way some path attributes are advertised from normal EBGP peering sessions.

- Local preferences is sent between sub-ASs
- The BGP next hop is <u>not</u> changed
- MED comparison between sub-ASs is allowed, with the assumption that the same MEDs mean the same thing among sub-ASs
- MED arithmetic (+/- MEDs) can be used for IGP-like metrics between sub-ASs

The IGP is still needed to carry internal routes for each sub-AS and the backbone links. In many cases, the IGP hierarchy could match with the sub-ASs, for example each sub-AS could be a separate IGP area. Or, you can even run different IGPs in sub-ASs.

It easy to apply routing policies between sub-ASs as well.  All of the path attributes that are needed for policies are available for use, and you gain EBGP-like functionality for peers that are internal to your network.  Confederations provides a quick-and-dirty way of integrating a network, as you can just make it a sub-AS in the short term without doing a rushed network redesign.  The sub-ASs can maintain autonomy in the confederation.  The one drawback is that migration from a full IBGP mesh to confederations is not simple.  While the CLI commands only consist of a few lines, lots of planning is required because each router in the network must be configured to know about the confederation AS and its sub-AS.

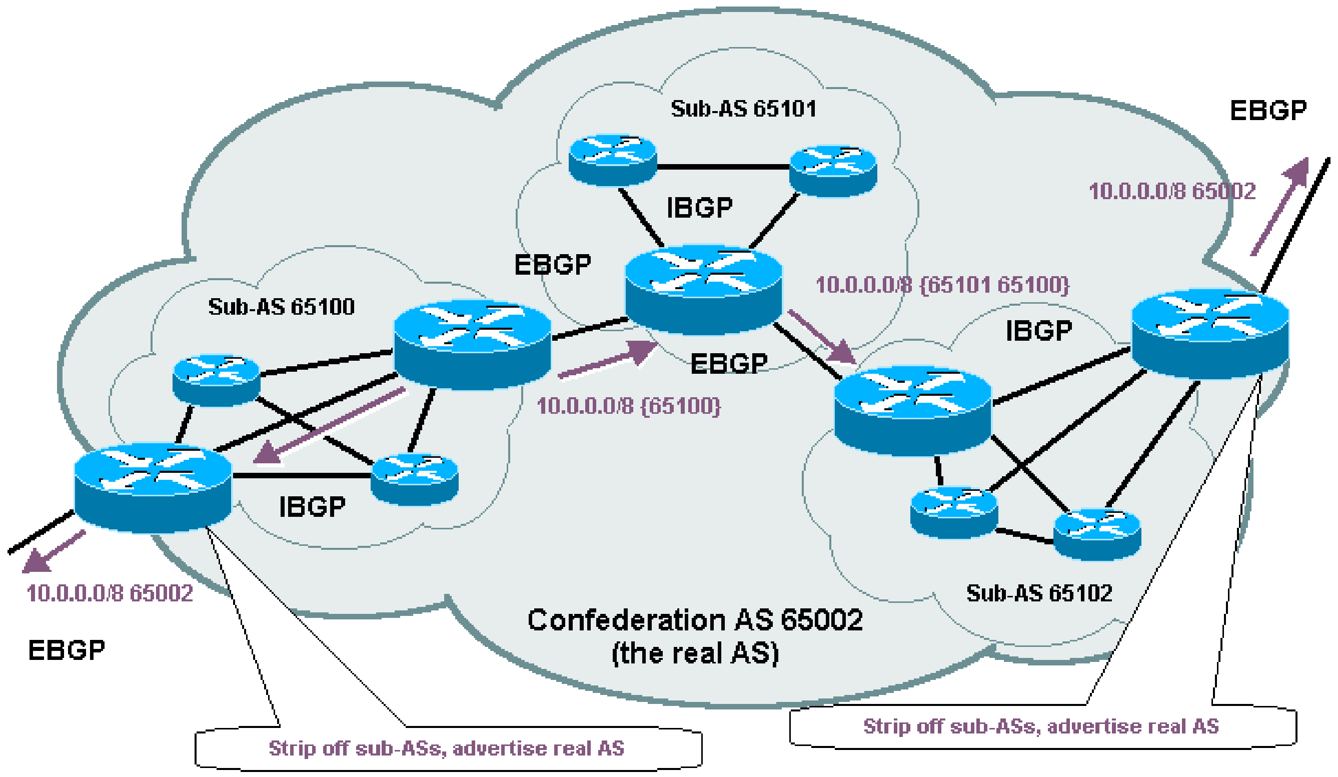
Two path attributes are extended to support confederations.

***AS_PATH* (Type Code 2, Well-known Mandatory - RFC 1771, extended in RFC 3065)**

Two new path segment types are defined to indicate that a sub-AS is in the AS path.  This prevents looping routing information within a sub-AS.  The same loop detection that is used for EBGP can also be applied to confederation peers.

- *AS_CONFED_SET* (path segment type 3): a unordered set of AS numbers, used to aggregate routes with different AS paths; denoted with "<>" in CLI output, for example: `<65001 65002> 65003`
- *AS_CONFED_SEQUENCE* (path segment type 4): an ordered set of sub-AS paths, from last advertised to origin sub-AS; denoted with "{}" in CLI output, for example: `{65001 65002} 65003`

These new segment types are similar to the *AS_SET* and *AS_SEQUENCE* types that are defined in RFC 1771, but are only used when confederations have been configured.



***COMMUNITIES* (Type Code 8, Optional Transitive - RFC 1991, extended in RFC 3065)**

A new well-known community is also defined for use in confederations.

- *NO_EXPORT_SUBCONFED (65535:65283)*: should not be advertised outside of the sub-AS, denoted with "`no-export-subconfed`" in the CLI

**Confederations Configuration**

Each router must be configured to be aware of the confederation AS and its sub-AS. Sub-ASs must be configured in each peer-group. For non-confederation peers, no changes are needed. This means that migration will be transparent to outside world.

The sub-AS is configured as each router's autonomous system setting

Syntax:
```
RS(config)# ip-router global set autonomous-system <sub-AS>
```

Example:
```
RS(config)# ip-router global set autonomous-system 65100
```

The confederation (real) AS is configured the same on all routers in the confederation.

Syntax:
```
RS(config)# ip-router global set confederation-id <real AS>
```

Example:
```
RS(config)# ip-router global set confederation-id 65001
```

The final step is to configure each group in the confederation as part of it. Peers that are not part of the confederation, such as transit or customer peer, should not have this option set.

Syntax:
```
RS(config)# bgp set peer-group <name> confederation
```

The following show commands display the confederation setting that are configured, and show the confederation path segments.

```
    RS# bgp show globals
     Router ID                 : 192.168.2.31
     Autonomous System         : 65100
     Confederation ID          : 65001
     BGP default preference    : 170
     BGP default localpref     : 100
     BGP default metric        : -1
     Action on a bad aspath    : Reset session
     Comparison of MED         : Only for same AS


    RS# bgp show routes all
    Local router ID is 192.168.2.31
    Status codes: > - best, * - valid, i - internal, t - stale
                  s - suppressed, d - damped
    Origin codes: i - IGP, e - EGP, ? - incomplete

        Network             Next Hop        Metric LocPrf Path
        -------             --------        ------ ------ ----
    *>  1/8                 172.16.0.1           0    100 64006 i
    *>  2/8                 172.16.0.1           0    100 64006 i
    *>  3/8                 172.16.0.1           0    100 64006 i
    *>  4/8                 172.16.0.1           0    100 64006 i
    *>  5/8                 172.16.0.1           0    100 64006 i
    *>  6/8                 172.16.0.9           0    100 60701 i
    *>  7/8                 172.16.0.9           0    100 60701 i
    *>  8/8                 172.16.0.9           0    100 60701 i
    *>  9/8                 172.16.0.9           0    100 60701 i
    *>  10.1/20             10.0.0.5            50    100 {65001} ?
    *>  10.2.1/24           192.168.2.36         0    100 60210 i
    *>  10.2.2/24           192.168.2.36         0    100 60220 i
    *>  10.2.3/24           192.168.2.36         0    100 60230 i
    *>  10.2.4/24           192.168.2.36         0    100 60240 i
    *>  10.3/20             10.0.0.5            50    100 {65001 65003} ?
    *>  11/8                172.16.0.9           0    100 60701 i
```

**Confederations Summary**

- Solves full IBGP mesh problem
- Route reflection can be used for even more scalability in a sub-AS
- Sub-AS autonomy through policies and IGP independence
- Migration from full IBGP mesh to confederations is non-trivial but can be done with planning
- Excellent case study is available at http://www.nanog.org/mtg-0006/confed.html

## Summary

The following table provides a summary of route reflection and confederations.

| Route Reflection | Confederations |
|---|---|
| Hierarchy approach | Divide-and-conquer approach |
| High scalability | Highest scalability |
| Easy migration | Harder migration |
| Easy expansion | Easy expansion |
| No autonomy among clusters | Autonomy among sub-ASs |
| Policy control among clusters | Policy control among sub-ASs |

# *Troubleshooting*

BGP tracing options are discussed and solutions to several common problems are provided in this section on troubleshooting BGP.

## Tracing

The CLI show commands that have been introduced in the previous sections provide the most information about problems that related to IP routing and what BGP is doing. These commands should be used first to gather information about a problem.

- `bgp show summary`: show a summary of all BGP peers
- `bgp show globals`: show global BGP configuration information
- `bgp show routes`: show BGP routes
- `bgp show neighbor`: show detailed information about a peer, including previous state and event
- `bgp show peer-host` <*IP address*> `advertised-routes`: show a peer's advertised routes
- `bgp show peer-host` <*IP address*> `received-routes`: show a peer's received routes
- `bgp show peer-host` <*IP address*> `all-received-routes:` show all of a peer's received routes, including ones rejected by a policy
- `bgp show regexp`: show routes matching an AS path regular expression
- `bgp show community`: show routes with a particular community
- `ip-router find route`: show details about the route to a particular destination

In some cases, for example when BGP peers are flapping continuously or when you suspect that something may be wrong at the protocol level, tracing the BGP messages may be the only way to find out more information about the problem. BGP tracing can decode and display BGP messages on the console for troubleshooting. Tracing will slow convergence because the CPU must decode and print each message on the console, and more verbose tracing will have a bigger impact. So, be sure to turn off tracing when you are done**.**

RapidOS support many flexible tracing option for BGP, which can be enabled for a peer, for a particular message type, for sent or received messages, or for everything related to BGP. Trace options are enabled in configuration mode with the following commands.

- `bgp trace packets`: trace all BGP packets
- `bgp trace open`: only trace OPEN messages
- `bgp trace update`: only trace UPDATE messages

Other more detailed options exist, for example state machine or timer tracing but you will probably never need to use them to solve most problems.

Trace commands share common options that can be used for more granularity or more verbose output.

- `detail`: enable full packet decode of messages

- `group`: enable tracing for all peers in a particular group
- `peer-host`: enable tracing for a particular peer
- `send`: enable tracing on all messages sent by the router
- `receive`: enable tracing on all messages received by the router

Once you have configured BGP tracing options, the following enable mode commands are used to toggle tracing in the routing engine. An optional trace duration can be specified, the default duration is 15 seconds.

Syntax:
```
RS# ip-router set trace-state on [trace-timer <seconds>|unlimited]
RS# ip-router set trace-state off
```

**BGP Tracing Example**

In this example, BGP will not come up with peer 10.0.0.2 for an unknown reason. Tracing is enabled to find out more about the problem.

Example:
```
RS(config)# bgp trace packets detail peer-host 10.0.0.2
RS# ip-router set trace-state on
```

Tracing output is only printed to the console by default, to enable tracing on a Telnet or SSH session use the following command.

```
RS# cli terminal monitor on
```

Sample tracing output is shown below.

```
    -12-19 16:02:21 BGP SEND 10.0.0.1 -> 10.0.0.2+32794
    -12-19 16:02:21 BGP SEND message type 1 (Open) length 39
    -12-19 16:02:21 BGP SEND version 4 as 1 holdtime 180 id 192.0.2.1 optional parameter length
    10
    -12-19 16:02:21 BGP SEND Optional parameter capabilities (10 bytes): MPCap: Inet Uni, Route
    Refresh:
    -12-19 16:02:21
    -12-19 16:02:21 BGP SEND 10.0.0.1 -> 10.0.0.2+32794
    -12-19 16:02:21 BGP SEND message type 4 (KeepAlive) length 19
    -12-19 16:02:21
    -12-19 16:02:21 BGP RECV 10.0.0.2+32794 -> 10.0.0.1
    -12-19 16:02:21 BGP RECV message type 3 (Notification) length 23
    -12-19 16:02:21 BGP RECV Notification code 2 (Open Message Error) subcode 2 (bad AS number)
    -12-19 16:02:21 BGP RECV Data (2 bytes): 00 01
```

The tracing output shows that a NOTIFICATION message was received with a “Bad AS Number” subcode. In this case, the AS number was configured with a different value than the peer was expecting. NOTIFICATION message often provide the most insight into peer establishment or flapping problems, because the message contains the exact cause of the error.

## Common Problems

**A BGP session is cycling between *Connect* and *Active* states.**

There is probably a problem with IP connectivity between peers, such as a physical link failure or IP routing problem.

Make sure that all links are working between peers, and that IGP has correct routes. Verify that you can ping the other peer.

**IBGP does not come up between loopback addresses.**

Make sure you have the `local-address` option set for the peer group on each router. Make sure that the IP address (loopback address) of the peer is in your IGP on each router. Verify that each router can ping the other router's loopback address using its loopback address as the source IP address using the following command.

```
RS# ping <peer address> sip <router's loopback address>
```

If you cannot ping loopback addresses, there is an IP routing problem in the network that is preventing IBGP from making its TCP connection.

**A route is received with BGP but is not selected as the active route.**

Verify that the route's next hop address is reachable, BGP must be able to resolve the BGP next hop address with an IP next hop address. Make sure there are no other RIB routes which have a higher preference, for example a static or IGP route.

**BGP peers are flapping.**

Enable tracing and look at the NOTIFICATION messages for exact details.

**How do I make sure my advertisements are propagating correctly over the Internet?**

CLI commands only show information about each router's peers, there is no way to see global propagation across the Internet. Looking Glasses are routers that are accessible to the public which are run by various ISPs and exchange points around the world to let you look at their BGP routing table. Two types exist, interactive ones which can be accessed with Telnet, and web-based one which can be access with a web browser. There are many Looking Glasses around the world, an extensive list can be found at the URL http://david.remote.net/connectivity/. A short list of popular ones follows below.

- telnet://ner-routes.bbnplanet.net/
- telnet://route-server.ip.att.net/
- telnet://route-views.oregon-ix.net/
- telnet://route-server.opentransit.net/
- http://nitrous.digex.net/
- http://lg.level3.net/

Feel free to login and look around, especially after making changes to BGP in your network. For grins, try the command "`show ip bgp | inc e`" to see who is advertising EGP origins.

---

# *Appendix*

## Summary of Supported Features

Riverstone Networks' implementation of BGP is based on GateD, with considerable modifications and enhancements. These enhancements provide a robust and rich set of BGP features on the Riverstone RS platform. The following list of BGP features is supported, and are described in more on the previous sections and in the product documentation.

- Flexible IBGP and EBGP peers and peer groups, with support for per-peer or per-group options which include:
    - Granular control on route redistribution policies by protocol type between autonomous systems
    - Default "soft" reconfiguration, with automatic evaluation of routing policies after changes
    - MD5 TCP header authentication
    - EBGP multihop
    - Next-hop-self setting
    - BGP update source (local address) setting
    - Damping control and tuning of suppress, reuse and decay parameters
    - Mminimum prefix mask import limit (i.e. do not accepts prefixes with a longer mask than /N)
    - Maximum prefix count import limit (i.e. only accept a maximum of N prefixes from a peer)
    - Private AS removal

- Route maps for fine-grained prefix control on import and export policies. These can be applied to a single peer, a peer group, or a route redistribution policy directly for maximum flexibility. Route maps support full prefix and path attribute matching and setting on attributes such as:
    - Origin
    - AS path, with POSIX regular expression matching and prepending
    - Next hop
    - MEDs, with addition and subtraction capability
    - Local preference
    - Communities and Extended Communities

- If desired, multiple route maps can be applied to a peer or peer group in a sequence, or route maps can be nested to create scalable and complex routing policies. Route maps can also be combined with the following lists for more granularity in policies:
    - AS path lists, with POSIX regular expression support
    - Community lists
    - Prefix lists which support exact, longer, or a mask range matching

- BGP policy accounting, which allows for traffic classification based on communities for billing or capacity planning. Please see the BGP Accounting Feature Application Note for details and example configurations.

- IBGP scaling features:
    - Route reflection, with multiple clusters

  - Confederations

- Global routing features:
  - Standards-based interoperability with core Internet routers such as Cisco and Juniper
  - Massive RIB, FIB and peer capacity
  - Multipath routing table with support for four equal cost routes per-protocol (OSPF, IS-IS, BGP, etc)
  - IGP metric to MED redistribution
  - Fine control of routing protocol metrics

## RFC/I-D Support by RapidOS Version

All BGP RFCs and I-Ds can be found on the IETF's IDR Charter page at http://www.ietf.org/html.charters/idr-charter.html.

| RapidOS Version | RFC/I-D | Title |
|---|---|---|
| 7.0.0.0 | RFC 1657 | Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2 |
| | RFC 1745 | BGP4/IDRP for IP---OSPF Interaction |
| | RFC 1771 | A Border Gateway Protocol 4 (BGP-4) |
| | RFC 1965 | Autonomous System Confederations for BGP |
| | RFC 1966 | BGP Route Reflection An alternative to full mesh IBGP |
| | RFC 1997 | BGP Communities Attribute |
| | RFC 2385 | Protection of BGP Sessions via the TCP MD5 Signature Option |
| | RFC 2439 | BGP Route Flap Damping |
| 8.0.0.0 | RFC 2842 | Capabilities Advertisement with BGP-4 |
| | RFC 2918 | Route Refresh Capability for BGP-4 |
| 9.0.0.0 | draft-ietf-idr-bgp-ext-communities-03.txt | BGP Extended Communities Attribute |
| | draft-ietf-idr-restart-03.txt | Graceful Restart Mechanism for BGP |

## References and Links

### Riverstone Network's White Papers

- BGP Accounting Feature Application Note
- BGP Functionality and Certification Test Plan
- Route Maps: Filtering on AS Numbers

### Riverstone Network's Documentation

- Configuration Database: a collection of short documents that quickly explain how to configure a specific feature by providing a diagram, the complete configurations, and commentary.
- Product Manuals

## Books

- Halabi, Bassam, and McPherson, Danny. *Internet Routing Architectures, Second Edition*. Indianapolis: Cisco Press, 2000. ISBN 157870233X.

  An excellent and in-depth book on BGP with a good introduction to the protocol as well as many simple and complex practical examples. Configurations are Cisco-specific but the principles apply to any vendor.

- Stewart, John W. *BGP4: Inter-Domain Routing in the Internet*. Reading: Addison Wesley Longman, 1999. ISBN 0201379511.

  This book presents a short and easy, vendor-neutral introduction and overview of BGP. It is great to keep handy as a BGP reference.

## Links

- http://david.remote.net/connectivity/, an excellent site with many networking links as well as an extensive list of Looking Glasses.

- http://www.nanog.org/mtg-0202/golding.html, a tutorial on designing and implementing routing policies
- http://www.as3257.net/html/communities.htm, has great examples of policies using communities
- http://www.nanog.org/mtg-0006/confed.html, a case study on migration to confederations that includes routing policy recommendations
- http://www.ripe.net/docs/ripe-210.html, a recommendation by RIPE on damping configurations
- http://www.ietf.org/internet-drafts/draft-manning-dsua-07.txt, a basic list of what prefixes to filter on AS borders

---

# ***Glossary***

## Autonomous System (AS)

A set of routers under a single technical administration, using an interior gateway protocol and common metrics to determine how to route packets within the AS, and using an exterior gateway protocol to determine how to route packets to other ASs. An AS looks like one entity to the outside world, even though there may be multiple networks within the AS. They are represented by numbers ranging from 1 – 65535. Numbers 64512 – 65535 are reserved for private use and should never appear in an Internet routing table.

## BGP Speaker

A router that sends, receives, and processes BGP messages. Also called a BGP peer or neighbor, these terms are used interchangeably and mean the same thing.

## Differences Between IGPs and EGPs

IGPs may have faster convergence than BGP, but simply will not scale to support the number of inter-domain routes that are needed for routing between ASs. IGPs also do not preserve the path attributes, which are essential for routing policies.  Separate protocols allow separation of interior, exterior and customer routes, which gives you stability and policy control.  You should never run your IGP with a customer (some exceptions apply for VRFs when providing VPN services).

## EBGP and IBGP

BGP can run in two modes that each has a very different behavior when advertising routing information.

- *EBGP*: external BGP runs between routers in different ASs
- *IBGP*: internal BGP runs between routers in the same AS

## Exterior Gateway Protocol (EGP)

A protocol used to communicate network reachability and routing information between autonomous systems.  Also called an *inter-domain* routing protocol.
Example: GGP, EGP (“EGP” is an EGP), BGP

## Interior Gateway Protocol (IGP)

A protocol used to communicate network reachability and routing information within an autonomous system.
Example: RIP, OSPF, IS-IS

## IP Prefix

A format for specifying an IP network or a group of IP networks that uses a 2-tuple representation of <length, prefix>.
Example: <8, 10> = 10.0.0.0/8
<16, 192.168> = 192.168.0.0/16

A longer mask means a more-specific prefix, and a shorter mask means a less-specific prefix.
Example: 10.0.0.0/8 is less-specific than 10.1.0.0/24
10.0.0.0/8 contains 10.1.0.0/24

## Network Layer Reachability Information (NLRI)

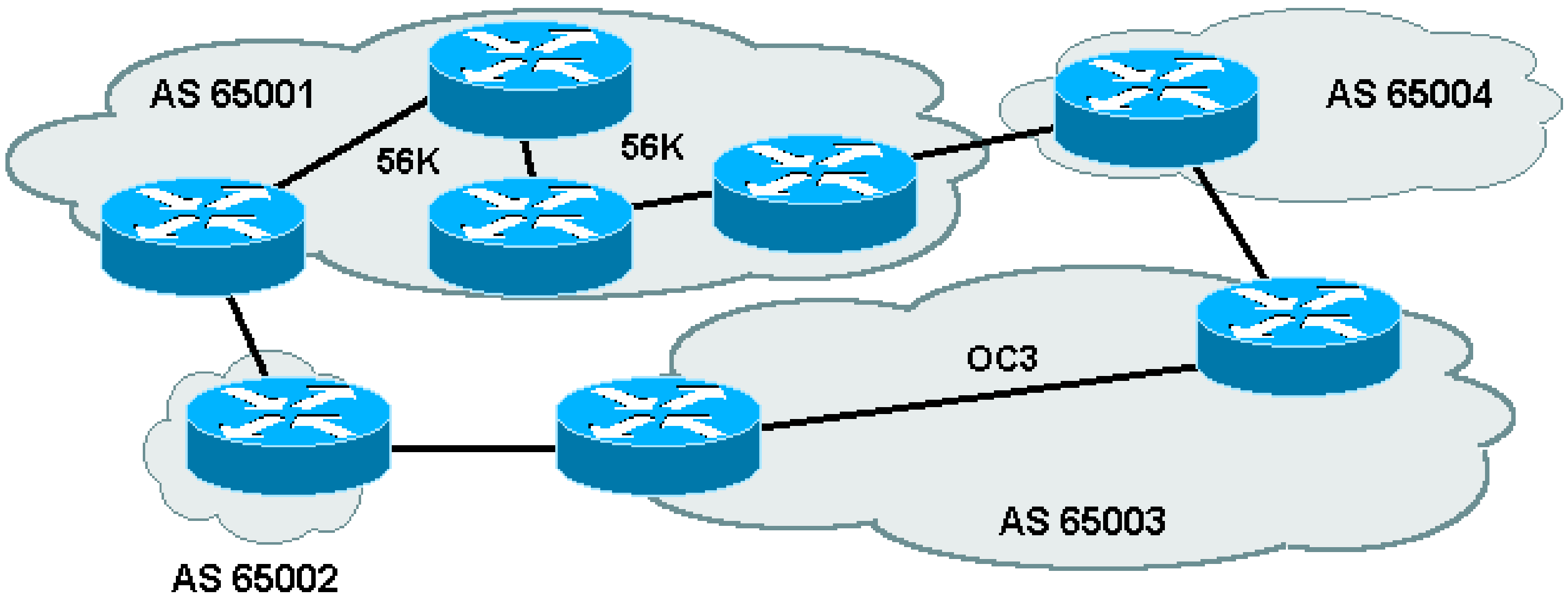
The NLRI lists the set of destinations which are advertised by a BGP speaker in an UPDATE message. It contains one or more 2-tuples of IP prefixes.

## Path Attribute

A set of parameters that describe various path characteristics of an IP prefix. Path attributes are used to select the best of multiple paths to the same destination. BGP routing policies make extensive use of path attributes in order to influence the way the best route is selected.
Example: AS Path, Next Hop, Local Preference, MED

## Path Vector Algorithm – RFC 1322

A path vector algorithm defines a route as a pairing between a destination and the attributes of the path to that destination. It considers multiple attributes of the path to the destination in order to choose the best route. Compare this with a distance vector algorithm which only uses a single distance metric to choose the best route. Path vector as it pertains to BGP means that BGP routing information carries a sequence of AS numbers that identifies the path that the prefix has traversed. If each AS only contained one router, then a path vector and distance vector algorithm could choose the same path.



BGP treats each AS equally when considering the path, no matter how big or small it is. BGP does not know how many routers or what type of links are in an AS. In the picture above, two paths from AS 65002 to AS 65004 look the same to BGP.

## Route Flap

A route flap is a route oscillation that occurs when a route is advertised and then withdrawn, or route is withdrawn and then readvertised in rapid succession. EBGP flapping causes global churn in the routing table, because the flap ripples across the Internet and each router must process the routing information change. IBGP flapping causes irregular traffic flow and reachability problems within the local AS, and can affect EBGP stability if IBGP routes are advertised to EBGP peers. Rapid flapping can consume significant CPU cycles that are spent on processing the routing updates. Route flapping usually indicates a problem, such as a circuit going up and down, or fatal recurring errors between BGP peers.

---