This chapter explores the various aspects of BGP policy control:

- Policy control techniques
- Conditional advertisement
- Aggregation and deaggregation
- Local AS
- QoS policy propagation
- BGP policy accounting
- Case study: AS integration via the Local AS

CHAPTER 4

# Effective BGP Policy Control

Throughout this book, you have learned that BGP is first and foremost a policy tool. This results in BGP's being used to build very complex policy-based architectures. The protocol itself provides a list of attributes through which you can set policies. Additionally, Cisco IOS software further expands and enhances what is available with additional tools and knobs. This chapter examines these tools and how you can use them to build complex and effective BGP policies.

## Policy Control Techniques

BGP employs many common policy control techniques. This section starts with regular expressions and then describes various forms of filter lists, route maps, and policy lists.

### Regular Expression

A *regular expression* is a formula for matching strings that follow a certain pattern. It evaluates text data and returns an answer of true or false. In other words, either the expression correctly describes the data, or it does not.

A regular expression is foremost a tool. For example, a regular expression can help extract the needed information from a large IOS output quickly, as shown in Example 4-1.

**Example 4-1** *Regular Expression to Extract All Neighbors' Maximum Data Segment Sizes*

```
R2#show ip bgp neighbors | include max data segment
Datagrams (max data segment is 1460 bytes):
Datagrams (max data segment is 1460 bytes):
Datagrams (max data segment is 1460 bytes):
```

As a formula, a regular expression allows pattern matching in BGP AS_PATH and community policy settings. Example 4-2 shows the use of a regular expression to describe an AS_PATH pattern that matches all AS_PATHs that are originated from the neighboring AS 100.

**Example 4-2** *Regular Expression Matches AS_PATH Patterns*

```
ip as-path access-list permit ^(100_)+$
```

## Components of a Regular Expression

A regular expression consists of two types of characters:

- Characters to be matched, or regular characters
- Control characters or metacharacters that have special meanings

To really make good use of regular expressions, it is critical to understand the control characters and how they are used. Control characters can be grouped into three types:

- **Atom characters, or atoms**—An atom is an independent control character or placeholder character that defines or expands the regular characters that are before or after it. Some atoms can be standalone, without regular characters.
- **Multiplier characters, or multipliers**—A multiplier follows an atom or a regular character and is used to describe repetitions of the character immediately before it. Except for the dot (.) character, all other atom characters must be grouped with regular characters before a multiplier is appended.
- **Range characters**—Range characters (brackets) specify a complete range.

Table 4-1 lists the common atom characters.

**Table 4-1** *Common Atom Characters and Their Usage*

| Atom Character | Usage |
|---|---|
| . | Matches any single character, including white space. |
| ^ | Matches the beginning character of a string. |
| $ | Matches the ending character of a string. |
| _ | Underscore. Matches a comma (,), left brace ({), right brace (}), the beginning of an input string, the end of an input string, or a space. |
| \| | Pipe. It is an OR, meaning that it matches either of two strings. |
| \ | An escape character to turn a control character that immediately follows into a regular character. |

Some simple examples are listed in Table 4-2.

**Table 4-2** *Examples of Atoms*

| Regular Expression | Usage |
|---|---|
| ^a.$ | Matches a string that begins with character a and ends with any single character, such as ab, ax, a., a!, a0, and so on. |
| ^100_ | Matches 100, 100 200, 100 300 400, and so on. |
| ^100$ | Matches 100 only. |
| ^100_500_ | Matches 100 500, 100 500 500, and so on. |
| 100$\|400$ | Matches 100, 2100, 100 400, 400, 100 100, 1039 2400, 600 400, and so on. |
| ^\(65000\)$ | Matches (65000) only. |

Table 4-3 shows the common multiplier characters.

**Table 4-3** *Multipliers and Their Usage*

| Multiplier | Usage |
|---|---|
| * | Any sequence of the preceding character (zero or more occurrences). |
| + | One or more sequences of the preceding character (one or more occurrences). |
| ? | Matches a preceding character with zero or one occurrences. |

A multiplier can be applied to a single-character pattern or a multicharacter pattern. To apply a multiplier to a multicharacter pattern, enclose the pattern in parentheses. Some simple examples are shown in Table 4-4.

**Table 4-4** *Examples of Multipliers*

| Regular Expression | Usage |
|---|---|
| abc*d | Matches abd, abcd, abccd, abcccd, and so on. |
| abc+d | Matches abcd, abccd, abcccd, and so on. |
| abc?d | Matches abd, abcd, abcdf, and so on. |
| a(bc)?d | Matches ad, abcd, cabcd, and so on. |
| a(b.)*d | Matches ad, ab0d, ab0b0d, abxd, abxbxd, and so on. |

The characters [ ] describe a range. Only one of the characters within the range is matched. You can make the matching exclusive by using the caret (^) character at the start of the range to exclude all the characters within the range. You can also specify a range by providing only the beginning and the ending characters separated by a dash (-). Some simple examples are shown in Table 4-5.

**Table 4-5** *Examples of Ranges*

| Regular Expression | Usage |
|---|---|
| [aeiouAEIOU] | Matches a, aa, Aa, eA, x2u, and so on. |
| [a-c1-2]$ | Matches a, a1, 62, 1b, xv2, and so on. |
| [^act]$ | Matches d, efg*, low2, actor, path, and so on, but not pact. |

## How to Use Regular Expressions in Cisco IOS Software

Regular expressions in IOS are only a subset of what is available from other operating systems. The use of regular expressions within IOS can be generally described in two categories:

- Filtering the command output
- Pattern matching to define policies

Regular expressions can be used in filtering outputs of **show** and **more** commands. The entire line is treated as one string. Table 4-6 shows the three types of filtering that can be done on an output.

**Table 4-6** *Regular Expressions Used to Perform Three Types of Output Filtering*

| Keyword | Usage |
|---|---|
| **begin** | Begins output lines with the first line that contains the regular expression. |
| **include** | Displays output lines that contain the regular expression. |
| **exclude** | Displays output lines that do not contain the regular expression. |

To filter the output, send the output with a pipe character (|) followed by the keyword and a regular expression. For example, **show run | begin router bgp** shows the part of the running configuration that begins with **router bgp**. To interrupt the filtered output, press **Ctrl-^** (press Ctrl, Shift, and 6 at the same time). Example 4-3 shows an example of filtering **show ip cef** output to show all the prefixes associated with the interface Ethernet0/0.

**Example 4-3** *Filtering* **show ip cef** *Output with a Regular Expression*

```
R1#show ip cef | include Ethernet0/0
172.16.0.0/16       192.168.12.2         Ethernet0/0
192.168.12.0/24     attached             Ethernet0/0
192.168.12.2/32     192.168.12.2         Ethernet0/0
192.168.23.0/24     192.168.12.2         Ethernet0/0
192.168.25.0/24     192.168.12.2         Ethernet0/0
192.168.36.0/24     192.168.12.2         Ethernet0/0
```

**NOTE** To type a question mark in a regular expression on the router, first press **Ctrl-V** (Escape for CLI), and then you can enter **?**.

Regular expressions are used extensively in pattern matching to define BGP policies, such as AS_PATH filtering. The AS_PATH attribute lists, in reverse order, the AS numbers, separated by blank spaces, that the prefix has traversed. You can use the command **show ip bgp regexp** to verify the result of the configured regular expressions.

Table 4-7 shows some examples of common AS_PATH pattern matching using regular expressions.

**Table 4-7** *Examples of AS_PATH Pattern Matching Using Regular Expressions*

| AS_PATH Pattern | Usage |
|---|---|
| .* | Matches all path information—for example, no filtering. |
| ^$ | Matches updates originated from the local AS. |
| ^200$ | Matches all paths that start and end with AS 200—that is, only updates originated and sent from AS 200 (no AS prepending and no intermediary). For example, this does not match 200 200. |
| _200$ | Matches all routes originated from AS 200, including those prepended with 200. |
| ^200 | Matches any updates received from the neighboring AS 200, such as 200, 200 100, 200 300 100, 2001, and so on. |
| _200_ | AS_PATH contains AS 200 (the prefix passed through AS 200 but not necessarily originated by or received directly from AS 200), such as 200, 200 100, 300 200 100, and so on. |
| ^100(_100)*(_400)*$ | Matches paths from AS 100 and its immediate neighbor AS 400, such as 100, 100 100, 100 400, 100 400 400, 100 100 100 400 400, and so on. |

## Filter Lists for Enforcing BGP Policies

Filter lists are used extensively in BGP to define policies. This section covers prefix lists, AS path lists, and community lists.

### Prefix Lists

Prefix lists are used to filter IP prefixes and can match both the prefix number and the prefix length. Compared to regular access lists, use of prefix lists provides higher performance (fewer CPU cycles).

**NOTE** Prefix lists cannot be used as packet filters.

A prefix list entry follows the same general format as an IP access control list (ACL). An IP prefix list consists of a name for the list, an action for the list (permit/deny), the prefix number, and the prefix length. Here is the basic format of an IP prefix list:

```
ip prefix-list name [seq seq] {deny | permit} prefix/length
```

**NOTE** A distribute list is another way to filter BGP routing updates. It uses access lists to define the rules and is mutually exclusive with the prefix list.

Any prefixes entered are automatically converted to match the length value entered. For example, entering 10.1.2.0/8 results in 10.0.0.0/8. Example 4-4 shows a simple example of matching 172.16.1.0/24. As with an access list, a deny-all entry is implied at the end of the list.

**Example 4-4** *Matching 172.16.1.0/24*

```
ip prefix-list out-1 permit 172.16.1.0/24
```

Optionally, a sequence number can be supplied for each entry. By default, the sequence numbers are automatically generated in increments of 5. They can be suppressed with the command **no ip prefix-list seq**. Entries are processed sequentially based on the sequence number. The use of sequence numbers offers flexibility when modifying a portion of a prefix list.

With the basic form of the prefix list, an exact match of both prefix number and prefix length is assumed. In Example 4-4, the prefix list matches only the prefix 172.16.1.0/24. The prefixes 172.16.1.128/25 and 172.16.1.0/25, for example, are not matched.

To match a range of prefixes and lengths, additional optional keywords are needed. When a range ends at /32, the greater-than-or-equal-to (**ge**) can be specified. The value of **ge** must be greater than the length value specified by prefix/length and not greater than 32. The range is assumed to be from the **ge** value to 32 if only the **ge** attribute is specified. If the range does not end at 32, another keyword, **le**, must be specified. The use of **le** is discussed later in this section.

**NOTE** A prefix consists of a prefix number and a prefix length. When a range is specified for a prefix list, the prefixes are matched for a range of prefix numbers and prefix lengths. For example, if a prefix list is **172.16.1.0/24 ge 25**, the matched range of the prefix numbers is 172.16.1.0 255.255.255.0 (representing a network mask in this case). The range of the matched prefix lengths falls between 25 and 32, inclusive. Thus, prefixes such as 172.16.1.128/25 and 172.16.1.0/30 are included. As another example, if the prefix list is **172.16.1.0/24 ge 27**, the matched range of the prefix numbers is still the same—that is, 172.16.1.0 255.255.255.0. The difference between the two is the range of the matched prefix lengths is smaller in the second example.

Example 4-5 shows an example of matching a portion of 172.16.0.0/16. Notice that the range is between /17 and /32, inclusive. Thus, the network 172.16.0.0/16 is excluded from the match. The legacy extended ACL version is also included for comparison.

**Example 4-5** *Matching a Portion of 172.16.0.0 255.255.0.0*

```
ip prefix-list range-1 permit 172.16.0.0/16 ge 17
!
access-list 100 permit ip 172.16.0.0 0.0.255.255 255.255.128.0 0.0.127.255
```

**NOTE** Standard ACLs do not consider prefix lengths. To filter classless routing updates, you can use extended ACLs. The source address, together with wildcard bits, specifies the prefix number. The field of destination address in an extended ACL is used to represent the actual netmask, and the field of destination wildcard bits is used to denote how the netmask should be interpreted. In other words, the fields of destination address and wildcard masks indicate the range's prefix lengths. The following are some examples.

This denies the prefix 172.16.0.0/24 only (not a range):

```
access-list 100 deny ip host 172.16.0.0 host 255.255.0.0
```

This permits 172.16.0.0 255.255.0.0 (the entire class B range):

```
access-list 100 permit ip 172.16.0.0 0.0.255.255 255.255.0.0 0.0.255.255
```

This denies any updates with lengths of 25 bits or longer:

```
access-list 100 deny ip any 255.255.255.128 0.0.0.127
```

Besides numbered ACLs, named extended IP ACLs can also be used for this purpose.

---

The range can also be specified by the less-than-or-equal-to (**le**) attribute, which goes from the length value specified by prefix/length to the **le** value, inclusive. Example 4-6 shows an example of matching the entire range of 172.16.0.0/16—that is, 172.16.0.0 255.255.0.0 using the regular mask or 172.16.0.0 0.0.255.255 using the inverted mask. If you want to specify a range that does not start from the length, you must specify another keyword, **ge**, as discussed next.

**Example 4-6** *Matching the Entire Class B Range of 172.16.0.0/16*

```
ip prefix-list range-2 permit 172.16.0.0/16 le 32
```

Example 4-7 shows another example. Both the prefix list and the ACL versions are shown.

**Example 4-7** *Matching 172.16.0.0 255.255.224.0*

```
ip prefix-list range-3 permit 172.16.0.0/19 le 32
!
access-list 100 permit ip 172.16.0.0 0.0.31.255 255.255.224.0 0.0.31.255
```

When both **ge** and **le** attributes are specified, the range goes from the **ge** value to the **le** value. A specified **ge** value and/or **le** value must satisfy the following condition:

length < ge value <= le value <= 32

The expanded prefix list format follows. Note that the **ge** attribute must be specified before the **le** value:

```
ip prefix-list name [seq #] deny | permit prefix/length [ge value] [le value]
```

Example 4-8 shows an example of using both **ge** and **le** attributes to match a portion of 172.16.1.0/24. The ACL version is also included.

**Example 4-8** *Matching a Portion of 172.16.1.0 255.255.255.0*

```
ip prefix-list range-3 permit 172.16.1.0/24 ge 25 le 31
!
access-list 100 permit ip 172.16.1.0 0.0.0.255 255.255.255.128 0.0.0.126
```

Note that 172.16.1.0/24 is not in the range, nor are all the /32s. The matched ranges include all the following prefixes:

- **Two /25s**—172.16.1.0/25, 172.16.1.128/25
- **Four /26s**—172.16.1.0/26, 172.16.1.64/26, ..., 172.16.1.192/26
- **Eight /27s**—172.16.1.0/27, 172.16.1.32/27, ..., 172.16.1.224/27
- **16 /28s**—172.16.1.0/28, 172.16.1.16/28, ..., 172.16.1.240/28
- **32 /29s**—172.16.1.0/29, 172.16.1.8/29, ..., 172.16.1.248/29
- **64 /30s**—172.16.1.0/30, 172.16.1.4/30, ..., 172.16.1.252/30
- **128 /31s**—172.16.1.0/31, 172.16.1.2/31, ..., 172.16.1.254/31

Table 4-8 shows more examples of prefix lists.

**Table 4-8** *Additional Examples of Prefix Lists*

| Prefix List | What It Matches |
|---|---|
| 0.0.0.0/0 | Default network |
| 0.0.0.0/0 le 32 | Any address that has a length between 0 and 32 bits, inclusive |

## AS Path Lists

AS path filters are used to filter the BGP AS_PATH attribute. The attribute pattern is defined by a regular expression string, either permitted or denied per the list's action. With regular expressions and AS path filters, you can build complex BGP policies.

The AS path list is defined by the **ip as-path access-list** command. The **access-list-number** is an integer from 1 to 500 that represents the list in the global configuration:

**ip as-path access-list** *access-list-number* {**permit** | **deny**} *as-regular-expression*

The filter can be applied in a BGP **neighbor** command using a filter list or in a route map (discussed in the later section "Route Maps"). Example 4-9 shows the use of an AS path filter to allow incoming routes from peer 192.168.1.1 that are only originated in AS 100.

**Example 4-9** *Path Filter to Permit Only Routes Originated from AS 100*

```
neighbor 192.168.1.1 filter-list 1 in
!
ip as-path access-list 1 permit _100$
```

## Community Lists

Community lists are used to identify and filter routes by their common community attributes. There are two forms of community lists: numbered and named. Within each category, there are also standard and expanded formats. A standard format allows actual community values or well-known constants, and an expanded format allows communities to be entered as a regular expression string. There is a limit of 100 for either format of the numbered lists (1 to 99 for the standard format and 100 to 199 for the expanded format), but named lists have no limit. The general formats are as follows:

- Standard numbered list:

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

- Expanded numbered list:

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

- Standard named list:

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

- Expanded named list:

```
ip community-list expanded list-name {permit | deny} regular-expression
```

By default, the *community-number* value is a 32-bit number between 1 and 4294967295. If you enter it in the *aa:nn* format (the new format), the resulting format is converted to a 32-bit number. If you enable the new format globally using **ip bgp-community new-format**, the new format is displayed. This change is immediate. Note that the format you choose is important, because the filtering using a regular expression in an expanded list can have different results for different formats.

**NOTE** The new community format splits the 32-bit number into two 16-bit numbers, *aa:nn*. Each number is expressed in decimal format. Typically, aa is used to represent an AS number, and *nn* is an arbitrary 16-bit number to denote a routing or administrative policy. Methods to design a coherent community-based policy are discussed in more detail in Chapter 9, "Service Provider Architecture."

One or more community numbers (separated by a space) can be entered per entry, or multiple entries can be entered per list number or name. When multiple communities are entered into the same entry, a match is found only when all communities match the condition—that is, an AND comparison. When multiple entries are entered for the same list

number or name, a match is found when any entry matches—that is, an OR comparison. Example 4-10 shows two forms of community lists.

**Example 4-10** *Two Ways of Entering Community Lists*

```
ip community-list 1 permit 100:1 100:2
ip community-list 2 permit 100:1
ip community-list 2 permit 100:2
```

With **list 1**, a match is found only when both community values of 100:1 and 100:2 are attached to a prefix. For **list 2**, a match is found if a prefix has a community with either 100:1 or 100:2 or both. Note that the rules stated here apply only to matching community values. They do not indicate whether a community is permitted or denied. For example, if the community list 2 in Example 4-10 is changed to deny 100:1 and 100:2 and to permit all other community values, a prefix with a community of 100:1 and 100:2 results in a match, and the prefix is denied.

**NOTE** To announce community settings to a peer, you must configure the command **neighbor send-community** for that peer. The result of this command is to send the peer with the communities permitted by the local outbound policies of the best paths.

Besides private communities, there are four well-known communities, as discussed in Chapter 2, "Understanding BGP Building Blocks"—**internet**, **no-export**, **local-as**, and **no-advertise**.

Community values for a prefix can be set or reset in two ways:

- Use a **set** clause within a route map to set a community value, to add a community value (**additive**), or to remove all community values:

  ```
  set community {community-value [additive]} | none
  ```

- Use a **set** clause within a route map to selectively remove some community values:

  ```
  set comm-list community-list-number delete
  ```

  This route map **set** command removes communities from the community attribute of an inbound or outbound update. Each community that matches the given community list is removed from the community attribute. When used with this command, each entry of a standard community list should list only one community.

**NOTE** When both the **set community** and **set comm-list delete** commands are configured in the same instance of a route map, the delete operation is performed before the set operation.

## Route Maps

A route map is a flexible and powerful way to set BGP policies. It can set and reset both prefixes and BGP attributes based on predefined conditions. A route map is often used to define policies toward a BGP peer or during route generation. A route map can filter updates based on prefix, AS_PATH, communities, metrics, next hop, ORIGIN, LOCAL_PREF, WEIGHT, and so on. A route map often uses policy control lists to define BGP policies.

A route map is a named group of filters consisting of one or more instances. Each instance is identified by a unique sequence number that determines the order of processing. Instances are applied sequentially. If a match is found, the rest of the route map is skipped. If the route map is finished without a match, a deny action is performed. When used in the **neighbor** command, only one route map per type per direction is allowed for each neighbor.

Within each instance, you can set conditions using the **match** clause and set actions using the **set** clause. Example 4-11 shows a simple route map named Set-comm, which resets communities to 200:100 when updates are originated from AS 100.

**Example 4-11** *Simple Route Map Example*

```
ip as-path access-list 1 permit _100$
!
route-map Set-comm permit 10
 match as-path 1
 set community 200:100
route-map Set-comm permit 20
```

The second instance (with sequence number 20) is important, because without it, all other updates that don't match the first instance are not accepted. When no match clause is specified under an instance, the result is to permit any. This instance basically means that no action should be taken for prefixes that do not match the conditions in the first instance.

**NOTE** The **deny** keyword in a route map is equivalent to a **no** keyword for other commands, but it does not necessarily indicate to deny something. The exact meaning depends on the route map's purpose. For example, if a route map is to suppress a route, **deny** is used to unsuppress that route. The same concept also applies to other forms of filtering of BGP prefixes and attributes.

There are two ways to match more than one condition. You can enter multiple conditions in the same **match** command or in different **match** commands. The processing rules are as follows:

- An OR function is performed between multiple **match** parameters defined in the same **match** command, regardless of the type of **match** commands.
- An OR function is performed when there are multiple **match** commands of the same type. Actually, IOS converts this form into the form discussed in the preceding bullet.
- An AND function is performed if there are multiple **match** commands of different types in the same route map instance.

Example 4-12 shows how the preceding rules work. The route map foo matches either community 100:1 or 100:2. With the route map foo2, a match is found only when the prefix and both communities are matched.

**Example 4-12** *Processing Example When Multiple Conditions Are Set with* **match** *Commands*

```
ip community-list 1 permit 100:1
ip community-list 2 permit 100:2
ip community-list 3 permit 100:1 100:2
!
ip prefix-list 1 seq 5 permit 13.0.0.0/8
!
route-map foo permit 10
 match community 1 2
!
route-map foo2 permit 10
 match ip address prefix-list 1
 match community 3
```

You can use a route map in the following BGP commands:

- **neighbor**
- **bgp dampening**
- **network**
- **redistribute**

Additionally, you can use route maps in various commands for specific purposes:

- **suppress-map**
- **unsuppress-map**
- **advertise-map**
- **inject-map**
- **exist-map**
- **non-exist-map**
- **table-map**

## Policy Lists

Complex route maps often have more than one match clause of different types. In a medium to large network, many of the same match clauses are reused repeatedly by different route maps. If the same sets of match clauses can be extracted from a route map, they can be reused by more than one route map or in different instances of the same route map. These independent match clauses are called *policy lists*.

A policy list is a subset of route maps that contains only match clauses. When a policy list is referenced in another route map, all the match clauses are evaluated and processed as if they were configured directly in the route map. Match clauses are configured in policy lists with permit or deny statements. The route map evaluates and processes each match clause and permits or denies routes based on the configuration in the referenced policy list.

A policy list is configured with the **ip policy-list** command and is referenced within another route map using the **match policy-list** command. Two or more policy lists can be referenced within a route map, and each entry can contain one or more policy lists. When multiple policy lists are configured in the same **match policy-list** command, it is an OR operation; when multiple **match policy-list** statements are configured, it is an AND operation. The policy lists and all other match and set options within a route map instance can coexist.

Example 4-13 shows a route map configuration using policy lists. Two policy lists are configured: as100 and as200. In as100, a match is found when both the AS path starts with AS 100 and the community is 300:105. In as200, a match is found when the AS path starts with AS 200 and the community is 300:105. With the route map foo, first a match is made to select the prefix to be 10.0.0.0/8, and then an OR operation is made for the two policy lists. The final action is to change the local preference to 105 for the updates that match.

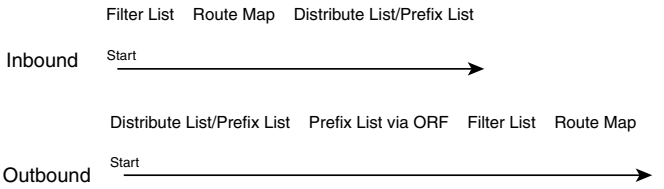
**Example 4-13** *Example of Policy List Configuration*

```
ip prefix-list 1 permit 10.0.0.0/8
ip as-path access-list 1 permit ^100_
ip as-path access-list 2 permit ^200_
ip community-list 1 permit 300:105
!
ip policy-list as100 permit
 match as-path 1
 match community 1
!
ip policy-list as200 permit
 match as-path 2
 match community 1
!
route-map foo permit 10
 match ip address prefix-list 1
 match policy-list as100 as200
 set local-preference 105
route-map foo permit 20
```

## Filter Processing Order

When multiple filters are configured per neighbor, each filter is processed in a specific order, as shown in Figure 4-1. For inbound updates, the filter list is processed first, followed by the route map. The distribute list or prefix list is processed last. On the outbound side, the distribute list or prefix list is processed first, and then the prefix list received via Outbound Route Filtering (ORF), and then the filter list. The route map is processed last.

**Figure 4-1** *Filter Processing Order*




An update has to pass through all the filters. One filter does not take precedence over another. If any filter does not match, the update is not permitted. For example, if an inbound update is permitted by the filter list and the route map but is denied by the prefix list, the update is denied. The same rule applies on the outbound side.

When a policy for a neighbor is configured in the **neighbor** command but the policy is not defined, the following are the default behaviors:

- For distribute lists and prefix lists, permit any.
- For filter lists and route maps, deny any.

# Conditional Advertisement

BGP by default advertises the permitted best paths in its BGP routing information base (RIB) to external peers. In certain cases, this might be undesirable. Advertisement of some routes might depend on the existence and nonexistence of some other routes. In other words, the advertisement is conditional.

In a multihomed network, some prefixes are to be advertised to one of the providers only if information from the other provider is missing, such as a failure in the peering session or partial reachability. The conditional BGP announcements are in addition to the normal announcements that a BGP router sends to its peers.

**NOTE** A conditional advertisement does not create routes; it only withholds them until the condition is met. These routes must already be present in the BGP RIB.

## Configurations

Conditional advertisement has two forms: advertisement of some prefixes when some other prefixes do not exist and advertisement of some prefixes when they do exist. The prefixes to be advertised are defined by a special route map called **advertise-map**. The condition is defined by a route map called **non-exist-map** for conditions that do not exist or by a route map called **exist-map** for conditions that do exist.

The first form of conditional advertisement is configured as follows:

```
neighbor advertise-map map1 non-exist-map map2
```

The route map associated with the non-exist-map specifies the prefix (or prefixes) that the BGP speaker tracks. Only permit is accepted; any deny is ignored. When a match is made, the status of the advertise-map is Withdraw; when no match is made, the status becomes Advertise.

Within the non-exist-map, a match statement for the prefix is required. You can configure it with a prefix list or a standard access list. Only an exact match is supported. Additionally, AS_PATH and community can be matched.

The route map associated with the advertise-map defines the prefix (or prefixes) that are advertised to the specific neighbor when the prefixes in the non-exist-map no longer exist—that is, when the status is Advertise. When the status is Withdraw, the prefix or prefixes defined in the advertise-map are not advertised or withdrawn. Note that the advertise-map applies only on the outbound direction, which is in addition to the other outbound filters.

The second form of conditional advertisement is configured as follows:

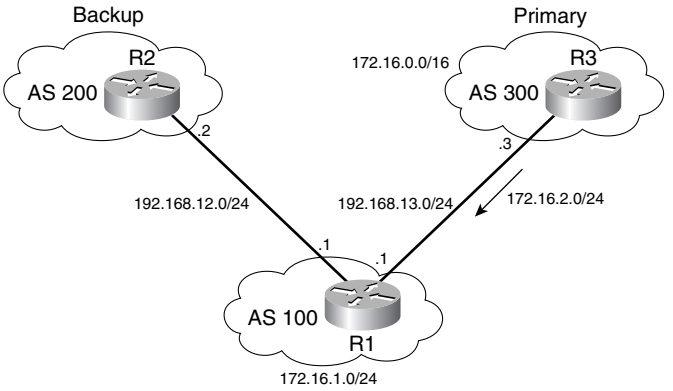
```
neighbor advertise-map map1 exist-map map2
```

In this case, the route map associated with the exist-map specifies the prefix (or prefixes) that the BGP speaker tracks. The status is Advertise when the match is positive—that is, when the tracked prefix exists. The status is Withdraw if the tracked prefix does not exist. The route map associated with the advertise-map defines the prefix (or prefixes) that are advertised to the specific neighbor when the prefix in the exist-map exists. Prefixes in both route maps must exist in the local BGP RIB.

## Examples

Figure 4-2 shows a topology of a conditional advertisement that tracks the nonexistence of a prefix. AS 100 is multihomed to AS 200 and AS 300, with the link to AS 300 as the primary connection. The address block of AS 100 is assigned from AS 300, within the range of 172.16.0.0/16. The address block 172.16.1.0/24 is not to be advertised to AS 200 unless the link to AS 300 fails. AS 300 sends 172.16.2.0/24 to AS 100, and it is tracked by the non-exist-map on R1. Example 4-14 shows R1's BGP configuration. Note that the community 100:300 is set and matched for the prefix to be tracked to ensure that the prefix is indeed from AS 300.

**Figure 4-2** *Conditional Advertisement in a Primary-Backup Scenario*




**Example 4-14** *Sample BGP Configuration for Conditional Advertisement on R1*

```
router bgp 100
 network 172.16.1.0 mask 255.255.255.0
 neighbor 192.168.12.2 remote-as 200
 neighbor 192.168.12.2 advertise-map AS200-out non-exist-map AS300-in
 neighbor 192.168.13.3 remote-as 300
 neighbor 192.168.13.3 route-map Set-comm in
!
ip community-list 1 permit 100:300
ip prefix-list AS300-track seq 5 permit 172.16.2.0/24
ip prefix-list Local-prefix seq 5 permit 172.16.1.0/24
!
route-map AS300-in permit 10
 match ip address prefix-list AS300-track
 match community 1
!
route-map Set-comm permit 10
 set community 100:300
!
route-map AS200-out permit 10
 match ip address prefix-list Local-prefix
```

When prefix 172.16.2.0/24 is present in R1's BGP RIB, 172.16.1.0/24 is not advertised to R2, as shown in Examples 4-15 and 4-16.

**Example 4-15** *Advertisement Status in R1 Under Normal Conditions*

```
R1#show ip bgp 172.16.2.0
BGP routing table entry for 172.16.2.0/24, version 3
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  192.168.12.2
  300
    192.168.13.3 from 192.168.13.3 (192.168.13.3)
      Origin IGP, metric 0, localpref 100, valid, external, best
      Community: 100:300

R1#show ip bgp neighbor 192.168.12.2 | include Condition-map
  Condition-map AS300-in, Advertise-map AS200-out, status: Withdraw
```

**Example 4-16** *R2 Does Not Have 172.16.1.0 Under Normal Conditions*

```
R2#show ip bgp 172.16.1.0
% Network not in table
```

When the session between R1 and R3 is down, 172.16.2.0/24 is removed from R1's BGP RIB. R2's advertisement status is now Advertise, as shown in Example 4-17. The prefix 172.16.1.0/24 is now available in R2, as shown in Example 4-18. For this design to work, it is important to ensure that the right prefix from the provider is being tracked.

**Example 4-17** *Advertisement Status During Primary Link Failure*

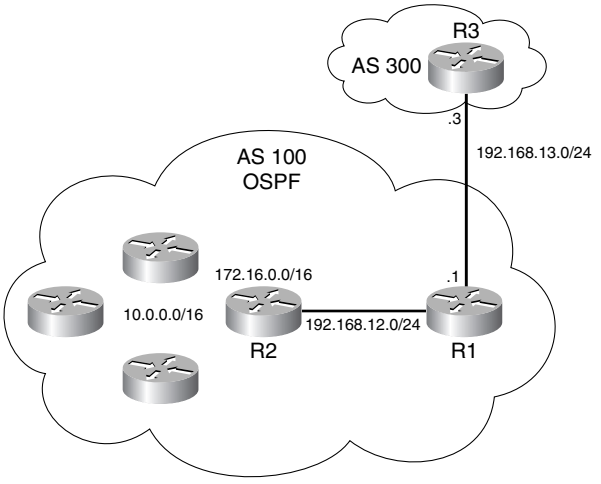
```
R1#show ip bgp neighbor 192.168.12.2 | include Condition-map
  Condition-map AS300-in, Advertise-map AS200-out, status: Advertise
```

**Example 4-18** *Prefix 172.16.1.0 Is Present on R2 During a Primary Link Failure*

```
R2#show ip bgp 172.16.1.0
BGP routing table entry for 172.16.1.0/24, version 14
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  100
    192.168.12.1 from 192.168.12.1 (192.168.13.1)
      Origin IGP, metric 0, localpref 100, valid, external, best
```

Figure 4-3 shows a topology of conditional advertisement to track the existence of a prefix. Within AS 100, R1 is the only BGP speaker, and it has an eBGP session with R3 in AS 300. All routers within AS 100 communicate using OSPF. The internal address block 10.0.0.0/16 is translated into a public block 172.16.0.0/16 on R2. The policy is that R1 should not advertise 172.16.0.0/16 to R3 unless 10.0.0.0/16 is available.

**Figure 4-3** *Conditional Advertisement to Track the Existence of a Prefix*




Example 4-19 shows a sample BGP configuration on R1. Both the prefix to be advertised (172.16.0.0) and the prefix tracked (10.0.0.0) are injected into the BGP RIB. The private prefix is then blocked from being advertised to R3 with the prefix list Block10. The exist map Prefix10 tracks the existence of 10.0.0.0/16, which is learned from OSPF. When the match returns true (status: Advertise), AS300-out is executed. When 10.0.0.0/16 is gone from OSPF (status: Withdraw), 172.16.0.0/16 is not advertised or withdrawn.

**Example 4-19** *Sample BGP Configuration on R1*

```
router bgp 100
 network 10.0.0.0 mask 255.255.0.0
 network 172.16.0.0
 neighbor 192.168.13.3 remote-as 300
 neighbor 192.168.13.3 prefix-list Block10 out
 neighbor 192.168.13.3 advertise-map AS300-out exist-map Prefix10
 no auto-summary
!
ip prefix-list Block10 seq 5 deny 10.0.0.0/16
ip prefix-list Block10 seq 10 permit 0.0.0.0/0 le 32
ip prefix-list adv-out seq 5 permit 172.16.0.0/16
ip prefix-list Private10 seq 5 permit 10.0.0.0/16
!
route-map Prefix10 permit 10
 match ip address prefix-list Private10
!
route-map AS300-out permit 10
 match ip address prefix-list adv-out
```

Example 4-20 shows what happens when 10.0.0.0/16 is available on R1's BGP RIB. The prefix 172.16.0.0/16 is advertised to R3.

**Example 4-20** *Advertisement of 172.16.0.0/16*

```
R1#show ip bgp 10.0.0.0
BGP routing table entry for 10.0.0.0/16, version 2
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  Local
    192.168.12.2 from 0.0.0.0 (192.168.13.1)
      Origin IGP, metric 20, localpref 100, weight 32768, valid, sourced, local,
  best

R1#show ip bgp 172.16.0.0
BGP routing table entry for 172.16.0.0/16, version 4
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  192.168.13.3
  Local
    192.168.12.2 from 0.0.0.0 (192.168.13.1)
      Origin IGP, metric 20, localpref 100, weight 32768, valid, sourced, local,
  best

R1#show ip route 10.0.0.0
Routing entry for 10.0.0.0/8, 1 known subnet
O E2    10.0.0.0/16 [110/20] via 192.168.12.2, 00:36:47, Ethernet0/0

R1#show ip bgp neighbor 192.168.13.3 | include Condition-map
  Condition-map Prefix10, Advertise-map AS300-out, status: Advertise

R3#show ip bgp 172.16.0.0
BGP routing table entry for 172.16.0.0/16, version 12
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  100
    192.168.13.1 from 192.168.13.1 (192.168.13.1)
      Origin IGP, metric 20, localpref 100, valid, external, best
```

Example 4-21 shows what happens when 10.0.0.0 is not available on R1.

**Example 4-21** *No Advertisement When 10.0.0.0 Is Down*

```
R1#show ip bgp neighbor 192.168.13.3 | include Condition-map
  Condition-map Prefix10, Advertise-map AS300-out, status: Withdraw

R3#show ip bgp 172.16.0.0
% Network not in table
```

Example 4-22 shows the output of **debug ip bgp update** on R1 when 10.0.0.0/16 is down. Example 4-23 shows a similar output when 10.0.0.0/16 is up again.

**Example 4-22** *Output of* **debug ip bgp update** *on R1 When 10.0.0.0 Is Down*

```
*Jul 29 21:37:39.411: BGP(0): route 10.0.0.0/16 down
*Jul 29 21:37:39.411: BGP(0): no valid path for 10.0.0.0/16
*Jul 29 21:37:39.411: BGP(0): nettable_walker 10.0.0.0/16 no best path
*Jul 29 21:37:39.411: BGP(0): 192.168.13.3 computing updates, afi 0, neighbor
  version 4, table version 5, starting at 0.0.0.0
*Jul 29 21:37:39.411: BGP(0): 192.168.13.3 update run completed, afi 0, ran for
  0ms, neighbor version 4, start version 5, throttled to 5
*Jul 29 21:38:20.331: BPG(0): Condition Prefix10 changes to Withdraw
*Jul 29 21:38:20.331: BGP(0): net 172.16.0.0/16 matches ADV MAP AS300-out: bump
  version to 6
*Jul 29 21:38:20.379: BGP(0): nettable_walker 172.16.0.0/16 route sourced locally
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 computing updates, afi 0, neighbor
  version 5, table version 6, starting at 0.0.0.0
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 172.16.0.0/16 matches advertise map
  AS300-out, state: Withdraw
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 send unreachable 172.16.0.0/16
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 send UPDATE 172.16.0.0/16 --
  unreachable
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 1 updates enqueued (average=26,
  maximum=26)
*Jul 29 21:38:20.379: BGP(0): 192.168.13.3 update run completed, afi 0, ran for
  0ms, neighbor version 5, start version 6, throttled to 6
```

**Example 4-23** *Output of* **debug ip bgp update** *on R1 When 10.0.0.0 Is Up*

```
*Jul 29 21:40:10.679: BGP(0): route 10.0.0.0/16 up
*Jul 29 21:40:10.679: BGP(0): nettable_walker 10.0.0.0/16 route sourced locally
*Jul 29 21:40:10.679: BGP(0): 192.168.13.3 computing updates, afi 0, neighbor
  version 6, table version 7, starting at 0.0.0.0
*Jul 29 21:40:10.679: BGP(0): 192.168.13.3 update run completed, afi 0, ran for
  0ms, neighbor version 6, start version 7, throttled to 7
*Jul 29 21:40:20.539: BPG(0): Condition Prefix10 changes to Advertise
*Jul 29 21:40:20.539: BGP(0): net 172.16.0.0/16 matches ADV MAP AS300-out: bump
  version to 8
*Jul 29 21:40:21.119: BGP(0): nettable_walker 172.16.0.0/16 route sourced locally
*Jul 29 21:40:37.639: BGP(0): 192.168.13.3 computing updates, afi 0, neighbor
  version 7, table version 8, starting at 0.0.0.0
*Jul 29 21:40:37.639: BGP(0): 192.168.13.3 172.16.0.0/16 matches advertise map
  AS300-out, state: Advertise
*Jul 29 21:40:37.639: BGP(0): 192.168.13.3 send UPDATE (format) 172.16.0.0/16,
  next 192.168.13.1, metric 20, path
*Jul 29 21:40:37.639: BGP(0): 192.168.13.3 1 updates enqueued (average=51,
  maximum=51)
*Jul 29 21:40:37.639: BGP(0): 192.168.13.3 update run completed, afi 0, ran for
  0ms, neighbor version 7, start version 8, throttled to 8
```

# Aggregation and Deaggregation

Aggregation of prefix information reduces the number of entries BGP has to carry and store. There are two common ways prefixes can be aggregated in BGP:

- Using the **network** command to enter an aggregate address and a static route to Null0
- Using the **aggregate-address** command to create an aggregate

Because the first method is straightforward, this section focuses on the second method—using the **aggregate-address** command. Here is the full command with its various options:

```
aggregate-address address mask [as-set] [summary-only] [suppress-map map1]
  [advertise-map map2] [attribute-map map3]
```

The creation of an aggregate in the BGP RIB is dependent on the existence of at least one component route in the local BGP RIB. Without any options specified, BGP attributes of the individual components are not included in the aggregate. The aggregate prefix has the following default attributes:

- **NEXT_HOP**—0.0.0.0 (local)
- **AS_PATH**—i (blank AS_PATH; origin code IGP)
- **MED**—Not set
- **LOCAL_PREF**—100
- **WEIGHT**—32768
- **AGGREGATOR**—Local
- **ATOMIC_AGGREGATE**—Tagged to the aggregate

By default, both the aggregate and its components are advertised. When **summary-only** is enabled for the aggregate, only the aggregate is advertised, and all the specific component routes are suppressed. The aggregate still maintains the default attributes just listed. If only a subset of the components are to be suppressed, you can define the subset with **suppress-map**. If a subset of suppressed routes needs to be made available, you can unsuppress those routes on a per-neighbor basis using the **neighbor unsuppress-map** command.

The option **as-set** allows AS path loop detection for the aggregate. Additionally, some of the attributes of components are included additively with the aggregate, even if they conflict. For example, if one component prefix has community set to 100:200 and another has it set to **no-export**, the community of the aggregate is 100:200 and **no-export**. The aggregate is not advertised to an eBGP peer.

The option **attribute-map** (a form of route map for setting BGP attributes) is used to clean up the aggregate's attributes. Using the previous community example, if an attribute map resets the community to 100:300, the previous two community values are replaced with 100:300, and the aggregate is advertised to an eBGP peer with 100:300. If only a subset of components are to be used to form the aggregate's attributes, these components can also be

defined by an **advertise-map**. Note that the aggregate's AS_SET is inherited only from the components that are defined in the map.

A common route aggregation practice is to group as large an address space as possible into as few prefix entries as possible. This is desirable in reducing the number of prefixes carried by the Internet, but it's detrimental to adjacent networks that have multiple connections to the aggregating network. One result of aggregation is that routing accuracy of neighbors is lost. In this situation, more-specific routes can be generated to better identify a prefix's address subsets across multiple connections. Deaggregation is a BGP feature that reconstructs components from a received aggregate prefix.

Deaggregation is accomplished by using the conditional injection feature. *Conditional injection* is the creation of more-specific components when an aggregate exists. These components are injected into the local BGP RIB to provide more-specific routing information in the local AS than the aggregate. These components can be installed in the IP RIB and advertised to other BGP peers within the AS.

Conditional route injection is configured as follows:

```
bgp inject-map map1 exist-map map2 [copy-attributes]
```

BGP tracks the prefix (the aggregate) in the exist-map to determine whether to install a prefix or prefixes as specified in the inject-map. The exist-map must have at least two match clauses:

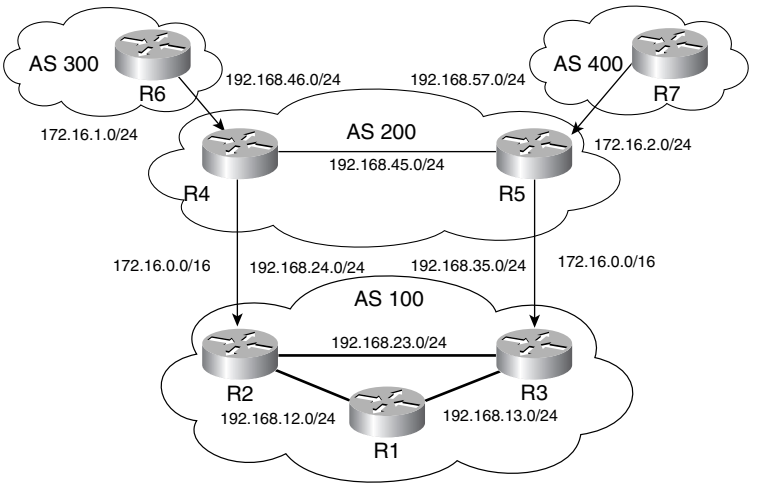
- **match ip address prefix-list** specifies the aggregate based on which to inject more specifics. Only one exact match is allowed.
- **match ip route-source** specifies the neighbor that sent the aggregate. The component inherits the attributes from the aggregate if the option **copy-attributes** is specified; otherwise, they are treated as locally generated routes for some of the attributes. The NEXT_HOP is always the eBGP peer that originated the aggregate. Additional matches can be made for AS_PATH and community.

Within the inject map, use **set ip address prefix-list** to define the prefixes to be injected into the local BGP RIB. The injected prefixes can be displayed with the **show ip bgp injected-path** command.

Figure 4-4 shows a sample topology that takes advantage of conditional injection to achieve deaggregation. Both AS 300 and AS 400 are customers of AS 200 and receive address blocks assigned by AS 200. The prefix block is 172.16.1.0/24 for AS 300 and 172.16.2.0/24 for AS 400. When announcing to AS 100, border routers of AS 200 summarize their address space to a single aggregate, 172.16.0.0/16.

Because AS 100 follows a best-exit policy (sometimes called *cold-potato routing*), it attempts to optimize its exit points. With a single aggregate, however, traffic destined for AS 300 might be exiting the AS via R3. If more-specific prefixes are available, you can control the traffic flows with better granularity.

**Figure 4-4** *Example of Conditional Injection*




With traffic statistics analysis, AS 100 determines that the best exit for 172.16.1.0/24 is via R2. It is also found that the best exit to 172.16.2.0/24 is via R3. In an effort to optimize the exit points, conditional injection is deployed on R2 and R3. The network address for each link is specified in Figure 4-4, with each router's number as the host address.

Example 4-24 shows a sample BGP configuration on R2. The route map **AS200-aggregate** matches the incoming aggregate from R4. If the match is positive, create 172.16.1.0/24 in the local BGP RIB. To prevent the injected routes from leaking back out, a community of **no-export** is set for the injected route. Also, a community of 100:200 is tagged for the route to indicate that it is a locally injected specific from AS 200.

**Example 4-24** *Sample BGP Configuration on R2*

```
router bgp 100
 bgp inject-map AS200-specific exist-map AS200-aggregate
 neighbor 192.168.12.1 remote-as 100
 neighbor 192.168.12.1 send-community
 neighbor 192.168.23.3 remote-as 100
 neighbor 192.168.23.3 send-community
 neighbor 192.168.24.4 remote-as 200
!
ip bgp-community new-format
ip prefix-list AS200-R4 seq 5 permit 192.168.24.4/32
ip prefix-list Aggregate seq 5 permit 172.16.0.0/16
ip prefix-list Specific seq 5 permit 172.16.1.0/24
!
route-map AS200-specific permit 10
```

**Example 4-24** *Sample BGP Configuration on R2 (Continued)*

```
 set ip address prefix-list Specific
 set community 100:200 no-export
!
route-map AS200-aggregate permit 10
 match ip address prefix-list Aggregate
 match ip route-source AS200-R4
```

Example 4-25 shows a similar configuration on R3. Another way to inject the specific components is to inject both specifics into routers R2 and R3 simultaneously. A preference can be set for one of the two.

**Example 4-25** *Sample BGP Configuration on R3*

```
router bgp 100
 bgp inject-map AS200-specific exist-map AS200-aggregate
 neighbor 192.168.13.1 remote-as 100
 neighbor 192.168.13.1 send-community
 neighbor 192.168.23.2 remote-as 100
 neighbor 192.168.23.2 send-community
 neighbor 192.168.35.5 remote-as 200
!
ip bgp-community new-format
ip prefix-list AS200-R5 seq 5 permit 192.168.35.5/32
ip prefix-list Aggregate seq 5 permit 172.16.0.0/16
ip prefix-list Specific seq 5 permit 172.16.2.0/24
!
route-map AS200-specific permit 10
 set ip address prefix-list Specific
 set community 100:200 no-export
!
route-map AS200-aggregate permit 10
 match ip address prefix-list Aggregate
 match ip route-source AS200-R5
```

Example 4-26 shows the BGP RIB on R1. Note that the BGP next hops are border routers that announce the aggregate and not the routers that inject the specifics. With the more-specific information, R1 directs traffic to R4 for 172.16.1.0 and to R5 for 172.16.2.0. The aggregate is used for all other traffic to 172.16.0.0.

**Example 4-26** *BGP RIB on R1*

```
R1#show ip bgp
BGP table version is 38, local router ID is 192.168.14.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete
```

*continues*

**Example 4-26** *BGP RIB on R1 (Continued)*

```
   Network          Next Hop            Metric LocPrf Weight Path
* i172.16.0.0       192.168.35.5                100      0 200 400 i
*>i                 192.168.24.4                100      0 200 300 i
*>i172.16.1.0/24    192.168.24.4                100      0 ?
*>i172.16.2.0/24    192.168.35.5                100      0 ?
```

Example 4-27 shows the BGP RIB on R2. Note that communities of 100:200 and **no-export** are attached to the injected prefixes.

**Example 4-27** *BGP RIB on R2*

```
R2#show ip bgp
BGP table version is 34, local router ID is 192.168.24.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.16.0.0       192.168.35.5                100      0 200 400 i
*>                  192.168.24.4                         0 200 300 i
*> 172.16.1.0/24    192.168.24.4                         0 ?
* i                 192.168.35.5                         0 ?
*>i172.16.2.0/24    192.168.35.5                100      0 ?

R2#show ip bgp 172.16.1.0
BGP routing table entry for 172.16.1.0/24, version 34
Paths: (2 available, best #1, table Default-IP-Routing-Table, not advertised to
  EBGP peer)
  Advertised to non peer-group peers:
  192.168.12.1 192.168.23.3
  Local, (aggregated by 200 192.168.46.4), (injected path from 172.16.0.0/16)
    192.168.24.4 from 192.168.24.4 (192.168.46.4)
      Origin incomplete, localpref 100, valid, external, best
      Community: 100:200 no-export
  Local, (aggregated by 200 192.168.57.5), (injected path from 172.16.0.0/16)
    192.168.35.5 (metric 20) from 192.168.23.3 (192.168.35.3)
      Origin incomplete, localpref 100, valid, internal
      Community: 100:200 no-export

R2#show ip bgp 172.16.2.0
BGP routing table entry for 172.16.2.0/24, version 32
Paths: (1 available, best #1, table Default-IP-Routing-Table, not advertised to
  EBGP peer)
  Not advertised to any peer
  Local, (aggregated by 200 192.168.57.5)
    192.168.35.5 (metric 20) from 192.168.23.3 (192.168.35.3)
      Origin incomplete, localpref 100, valid, internal, best
      Community: 100:200 no-export
```

When the link between R2 and R4 is down, the aggregate from R4 is removed. Under this condition, R2 stops the injection of the prefix 172.16.1.0/24. This is shown in the BGP RIB on R1 in Example 4-28. When the link between R3 and R5 is down as well, both 172.16.0.0 and 172.16.2.0 are also removed from AS 100 (not shown).

**Example 4-28** *BGP RIB on R1 When the Link Between R2 and R4 Is Down*

```
R1#show ip bgp
BGP table version is 56, local router ID is 192.168.14.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*>i172.16.0.0       192.168.35.5                100      0 200 400 i
*>i172.16.2.0/24    192.168.35.5                100      0 200 400 i
```

# Local AS

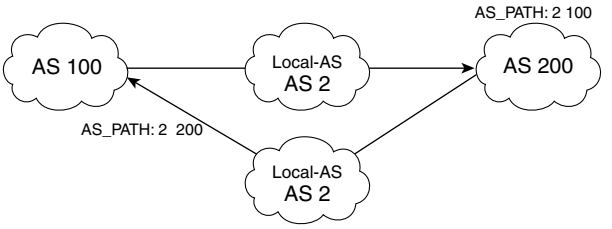
When two ISPs merge their networks, many challenges related to BGP design arise. When one AS is being replaced by another AS, its former peering autonomous systems might not honor the new AS and might continue to insist on the previous peering agreements. For example, if ISP A has a private peering agreement with ISP B, and if ISP A is acquired by ISP C, ISP B might not want to peer with ISP C but might honor the previous peering agreement with ISP A.

An ISP generally has various peering agreements with other ISPs. Changing the AS number on a large scale might be too disruptive to its peering sessions with other ISPs. Also, changing the AS number on all the routers in one large AS during one maintenance window might not be feasible or recommended. During the migration, both autonomous systems must coexist and continue to communicate. The BGP Local AS feature helps reduce these challenges.

With the Local AS feature, a BGP speaker can be physically in one AS and acts as such to some neighbors while it appears to be another AS to other neighbors. When sending and receiving AS_PATH to and from neighbors with Local AS configured, BGP prepends the Local AS to the real AS. For these neighbors, BGP uses the Local AS as the remote AS in the configuration. Thus, the Local AS number appears as if it were another AS inserted between the two real autonomous systems.

Figure 4-5 shows an example. When AS 2 is configured on AS 200 as a Local AS, the AS_PATH is prepended with AS 2 for updates from AS 100. When AS 100 receives updates from AS 200, the AS_PATH is prepended with AS 2.

**Figure 4-5** *AS_PATH Updates with Local AS*




**NOTE** Local AS can be used together with peer groups, but it cannot be customized for individual peers in a peer group. Local AS cannot have the local BGP AS number or the remote peer's AS number. The **local-as** command is valid only if the peer is a true eBGP peer. It does not work for two peers in different member autonomous systems in a confederation.

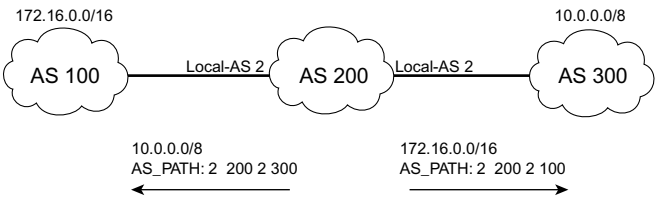
Example 4-29 shows a sample BGP configuration in AS 200 border routers for Figure 4-5. 192.168.1.1 is the IP address of a BGP speaker in AS 100. On 192.168.1.1, 2 instead of 200 is configured as the remote AS (not shown).

**Example 4-29** *Sample Local AS Configuration in AS 200*

```
router bgp 200
 neighbor 192.168.1.1 remote-as 100
 neighbor 192.168.1.1 local-as 2
```

Figure 4-6 shows another example of Local AS. In this case, AS 200 is configured with Local AS with two remote autonomous systems, AS 100 and AS 300. When AS 200 border routers advertise prefix 172.16.0.0/16 to AS 300, the AS_PATH is 2 200 2 100. Because loop detection is done only for incoming updates from an eBGP peer, this AS_PATH is not considered a condition of a loop. AS 300 accepts the prefix because it does not detect any loop of AS 300. Similarly, AS 100 accepts prefix 10.0.0.0/8. Multiple occurrences of the Local AS number in the eBGP updates indicate more than one point of Local AS sessions.

**Figure 4-6** *Local AS in Two Connections*

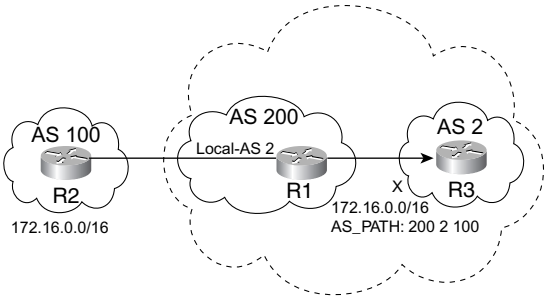


**NOTE** When Local AS is used, the AS_PATH length becomes longer. If AS_PATH length is used as a deciding factor in selecting preference, AS_PATH prepending might be needed on other paths so that path selection is not affected.

During AS migration, it is possible that some routers are in the original AS and others are in the new AS. When a border router is migrated to the new AS and is configured with Local AS to remote peers, the updates from this border router to other routers that are still in the old AS are denied, because the other routers detect an AS_PATH loop.

Figure 4-7 shows what happens. Before the migration, both R1 and R3 are in AS 2. When R1 is migrated to AS 200 (the new AS), the Local AS is configured with R2 in AS 100. When R3 receives the prefix 172.16.0.0/16, it detects its own AS in the AS_PATH, and the update is denied. Example 4-30 shows the output of **debug ip bgp update** in on R3.

**Figure 4-7** *Updates Denied on R3 with Local AS on R1*




**Example 4-30** *Loop Detection on R3 as Captured by* **debug ip bgp update in**

```
*Apr 22 04:59:32.563 UTC: BGP(0): 192.168.13.1 rcv UPDATE w/ attr: nexthop
  192.168.13.1, origin i, originator 0.0.0.0, path 200 2 100, community , extended
  community
*Apr 22 04:59:32.563 UTC: BGP(0): 192.168.13.1 rcv UPDATE about 172.16.0.0/16 --
  DENIED due to: AS-PATH contains our own AS;
```

As mentioned previously, loop detection is performed on the inbound of an eBGP session. Because the session between R1 and R3 is now eBGP, this detection is enforced.

The solution to the problem is to add the **no-prepend** option to the **local-as** command. With this option, R1 does not prepend its Local AS number to the update received from R2. For this example, the AS_PATH to R3 is then 200 100. The update is acceptable to R3. The case study near the end of this chapter provides a more-detailed discussion of how to migrate an AS using the Local AS feature.

# QoS Policy Propagation

Cisco Express Forwarding (CEF) and the forwarding information base (FIB) were discussed in Chapter 2. A FIB leaf has three policy parameters:

- Precedence
- QoS-group ID
- Traffic index

All three parameters can be used to provide differential treatment to an IP packet in forwarding or accounting. The precedence is as defined in the IPv4 header. After it is reset in IP packets, it can influence QoS treatment in other routers. The other two parameters are used by the local router only to differentiate traffic.

BGP can set these parameters when certain BGP prefixes and attributes are matched. With this information in CEF, policies can be created and accounted. Policy accounting using BGP is discussed in the section "BGP Policy Accounting."

QoS Policy Propagation via BGP (QPPB) lets you map BGP prefixes and attributes to CEF parameters that can be used to enforce traffic policing. Compared to other QoS methods, QPPB allows BGP policy set in one location of the network to be propagated via BGP to other parts of the network, where appropriate QoS policies can be created.

Configuring QPPB generally involves the following steps:

**Step 1** Identify BGP prefixes that require preferential treatment, and tag them with appropriate BGP attributes.

**Step 2** Set appropriate FIB policy parameters for each type of traffic.

**Step 3** Configure FIB address lookups for the tagged prefixes as packets are received on an interface, and set appropriate QoS policies.

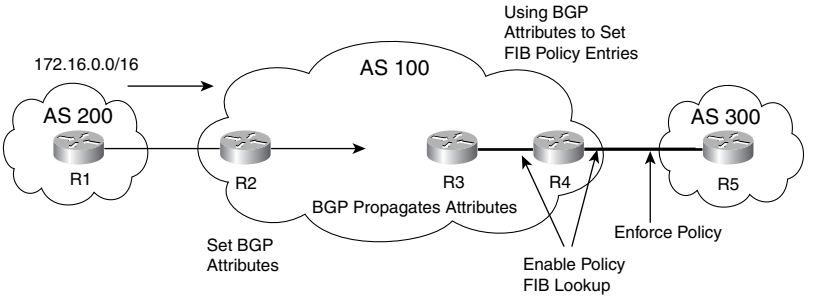
**Step 4** Enforce policing based on the lookups and settings done in Step 3 for packets received or transmitted.

The following sections describe each step in greater detail. Configuration examples appear later.

## Identifying and Tagging BGP Prefixes That Require Preferential Treatment

Figure 4-8 shows how this process works. Assume that AS 100 wants to create a special forwarding policy for traffic between AS 200 and AS 300 for prefix 172.16.0.0/16. When the prefix is first received from R1 via BGP, R2 tags the prefix with special BGP attributes, such as a specific community value.

**Figure 4-8** *How QoS Policy Propagation via BGP Works*




## Setting FIB Policy Entries Based on BGP Tagging

As the prefix is propagated via BGP inside AS 100 to R4, the attributes are propagated as well. When R4 receives the prefix with the matching attributes, it can set various FIB policy entries using the **table-map** command in BGP. For QPPB, either or both Precedence and QoS-group ID (a parameter internal to the router) can be set. The Precedence can have eight values, 0 to 7, and the QoS-group ID can have 99 values, 1 to 99. Each value or a combination of both values can represent one class of traffic. Note that these settings have no impact on traffic forwarding until they are used to classify and police the traffic (as discussed next).

**NOTE** Changes to the FIB/RIB tables are made when the IP RIB is cleared using **clear ip route ***, the BGP session is reset, or a router is reloaded. All of these actions can be disruptive to the traffic.

Within the FIB entry for the prefix 172.16.0.0/16, the following mappings are possible, depending on the table map configuration:

- 172.16.0.0 Precedence
- 172.16.0.0 QoS-group ID
- 172.16.0.0 Precedence and QoS-group ID

## Configuring Traffic Lookup on an Interface and Setting QoS Policies

The next step is to classify the incoming traffic from an interface based on the FIB policy entries. The definition of the incoming interface depends on the traffic's direction. If traffic is destined for 172.16.0.0/16 from AS 300, the incoming interface is the link between R4 and R5; if the traffic is destined for AS 300 from 172.16.0.0/16 (the return traffic), the incoming interface is the link between R3 and R4. On the incoming interfaces on R4, enable FIB policy lookup using the following command:

```
bgp-policy {source | destination} {ip-prec-map | ip-qos-map}
```

The keywords **source** and **destination** indicate whether to use the source or the destination IP address of an incoming packet to look up the FIB entries. On the link between R4 and R5, the incoming traffic is destined for 172.16.0.0/16, so you should use **destination**. On the link between R3 and R4, the incoming traffic is sourced from 172.16.0.0/16, so you should use **source**.

With this configuration command, appropriate QoS policies are also set if there is a match for both the address and QoS parameters. The interface map keyword specifies which of the two policy FIB entries to set for the packet. If **ip-prec-map** is specified, the IP precedence bits are set for the matching packets; if **ip-qos-map** is specified, the QoS-group ID is set. Note that setting IP precedence bits here might affect the QoS treatment of these packets on other routers.

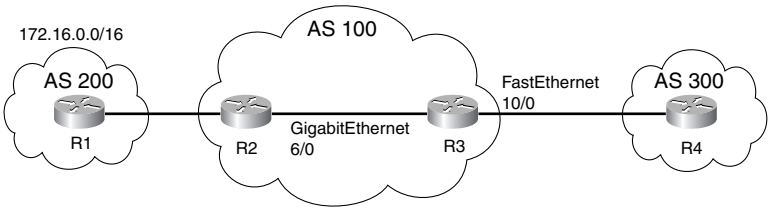
## Enforcing Policing on an Interface as Traffic Is Received and Transmitted

The last step of QPPB configuration is to create traffic policing on the interface to AS 300. This can be accomplished by using Committed Access Rate (CAR) and Weighted Random Early Detection (WRED). The policing can be done on the input to the router for traffic destined for 172.16.0.0 or on the output from the router for the return traffic sourced from 172.16.0.0. The policing is created based on the result of the policy lookup and settings done previously.

## An Example of QPPB

Figure 4-9 shows a simple topology that demonstrates how to configure QPPB. Within AS 100, special treatment is needed for traffic between AS 200 and AS 300 to and from the prefix 172.16.0.0/16. On R2, prefix 172.16.0.0/16 from R1 is tagged with a community of 100:200, and the prefix is propagated to R3 via iBGP. The FastEthernet 10/0 interface on R3 is used to demonstrate how QoS policing can be set for traffic destined for 172.16.0.0/16.

**Figure 4-9** *Example of QoS Policy Propagation*




Example 4-31 shows a sample BGP configuration on R3. The router number is used as the host address. The route map Set-policy sets the FIB QoS-group ID to 2 for prefixes matching the community 100:200, which is tagged for 172.16.0.0/16 by R2.

**Example 4-31** *Sample BGP Configuration on R3*

```
router bgp 100
 table-map Set-policy
 neighbor 192.168.23.2 remote-as 100
 neighbor 192.168.34.4 remote-as 300
!
ip community-list 1 permit 100:200
!
route-map Set-policy permit 10
 match community 1
 set ip qos-group 2
```

Examples 4-32 and 4-33 show the IP RIB and FIB entries, respectively. Note that prefix 172.16.0.0/16 is now set with qos-group 2.

**Example 4-32** *IP RIB Entry for 172.16.0.0*

```
R3#show ip route 172.16.0.0
Routing entry for 172.16.0.0/16
  Known via "bgp 200", distance 200, metric 0, qos-group 2, type internal
  Last update from 192.168.23.2 00:32:34 ago
  Routing Descriptor Blocks:
  * 192.168.23.2, from 192.168.23.2, 00:32:34 ago
      Route metric is 0, traffic share count is 1
      AS Hops 1, BGP network version 0
```

**Example 4-33** *FIB Entry for 172.16.0.0*

```
R3#show ip cef 172.16.0.0
172.16.0.0/16, version 23, cached adjacency 192.168.12.2
0 packets, 0 bytes, qos-group 2
  via 192.168.23.2, 0 dependencies, recursive
    next hop 192.168.23.2, GigabitEthernet6/0 via 192.168.23.2/32
    valid cached adjacency
```

To enable FIB lookup for the traffic destined for 172.16.0.0/16, policy lookup is enabled on the interface of FastEthernet 10/0. The keyword **destination** is used in the command. If there is a match for the destination address, a check is made into the FIB to determine if there are any matching QoS entries. In this example, ip-qos-map is configured for the interface, and QoS-group ID is set to 2 in FIB, which means that the QoS-group ID can be used to set QoS policies. An input CAR is configured for traffic matching a QoS-group ID of 2. A sample configuration is shown in Example 4-34.

**Example 4-34** *Sample Interface Configuration for QPPB*

```
interface FastEthernet10/0
 ip address 192.168.34.3 255.255.255.0
 no ip directed-broadcast
 bgp-policy destination ip-qos-map
 rate-limit input qos-group 2 5000000 4000 8000 conform-action transmit
  exceed-action drop
```

Example 4-35 shows the IP interface status. Example 4-36 shows traffic policing using CAR. A similar configuration can be made for the traffic sourced from 172.16.0.0 to AS 300 (not shown). The incoming interface then is GigabitEthernet 6/0. An outbound CAR should be configured on the interface of FastEthernet 10/0 to enforce the QoS policy.

**Example 4-35** *IP Interface Status of FastEthernet 10/0*

```
R3#show ip interface FastEthernet 10/0 | include BGP
  BGP Policy Mapping is enabled (output ip-qos-map)
```

**Example 4-36** *Interface CAR Status*

```
R3#show interface FastEthernet 10/0 rate-limit
FastEthernet10/0
  Input
    matches: qos-group 2
      params:  5000000 bps, 4000 limit, 8000 extended limit
      conformed 112 packets, 168448 bytes; action: transmit
```

**Example 4-36** *Interface CAR Status (Continued)*

```
      exceeded 0 packets, 0 bytes; action: drop
      last packet: 1300ms ago, current burst: 0 bytes
      last cleared 00:13:15 ago, conformed 1694 bps, exceeded 0 bps
```

# BGP Policy Accounting

BGP policy accounting (BPA) is another BGP feature that takes advantage of the FIB policy parameters. In this case, the parameter is traffic index. Traffic index is a router internal counter within a FIB leaf with values between 1 and 8. Think of the traffic index as a table of eight independent buckets. Each can account for one type of traffic matching certain criteria. The number of packets and bytes in each bucket of an interface is recorded.

You can use this feature to account for IP traffic differentially on an edge router by assigning counters based on BGP prefixes and attributes on a per-input interface basis.

Configuration of BPA generally involves the following steps:

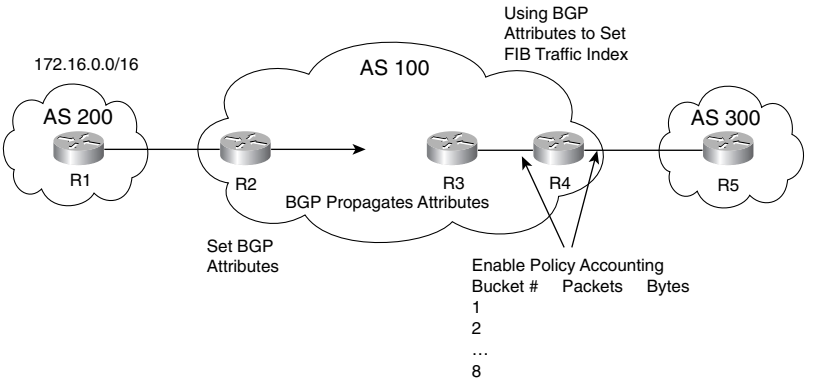
**Step 1** Identify BGP prefixes that require preferential treatment and tag them with appropriate BGP attributes.

**Step 2** Set a FIB traffic index for each type of traffic.

**Step 3** Enable BPA on an incoming interface.

Figure 4-10 shows how BGP policy accounting works. As prefix 172.16.0.0/16 is propagated from AS 200 to AS 300, certain BGP attributes are modified. On R4, a traffic index number can be set when a match is made for the attributes using the **table-map** command. A total of eight traffic classes can be accounted.

**Figure 4-10** *How BGP Policy Accounting Works*



**NOTE** Remember that changes to the FIB/RIB tables are updated when the IP RIB is cleared using **clear ip route ***, the BGP session is reset, or a router is reloaded. All these actions can be disruptive to the traffic.

On each incoming interface, you can enable policy accounting by using the command **bgp-policy accounting**. With this command, using destination IP addresses, traffic matching the criteria is accounted for in its respective bucket. The **show cef interface policy-statistics** command displays the per-interface table of traffic counters. The counters can be cleared using the **clear cef interface policy-statistics** command.

Using the topology shown in Figure 4-9, an example of BGP policy accounting is demonstrated here. For the prefix 172.16.0.0/16, the BGP community is set as before. On R3, a route map is created to update the FIB traffic-index, as shown in Example 4-37.

**Example 4-37** *Sample BGP Configuration on R3*

```
router bgp 100
 table-map Set-policy
 neighbor 192.168.23.2 remote-as 100
 neighbor 192.168.34.4 remote-as 300
!
ip community-list 1 permit 100:200
!
route-map Set-policy permit 10
 match community 1
 set traffic-index 1
```

The updated FIB for the prefix is shown in Example 4-38. To account for the prefix, policy accounting is enabled on FastEthernet 10/0. This is the incoming interface for traffic destined for 172.16.0.0. Note that this interface doesn't account for the return traffic, because the matching is done on the destination address. To account for the return traffic, policy accounting must be enabled on GigabitEthernet 6/0, and appropriate criteria must be set using the addresses of AS 300. Example 4-39 shows the accounting statistics on FastEthernet 10/0.

**Example 4-38** *FIB Traffic Index for 172.16.0.0*

```
R3#show ip cef 172.16.0.0
172.16.0.0/16, version 23, cached adjacency 192.168.23.2
0 packets, 0 bytes, traffic_index 1
  via 192.168.23.2, 0 dependencies, recursive
    next hop 192.168.23.2, GigabitEthernet6/0 via 192.168.23.2/32
    valid cached adjacency
```

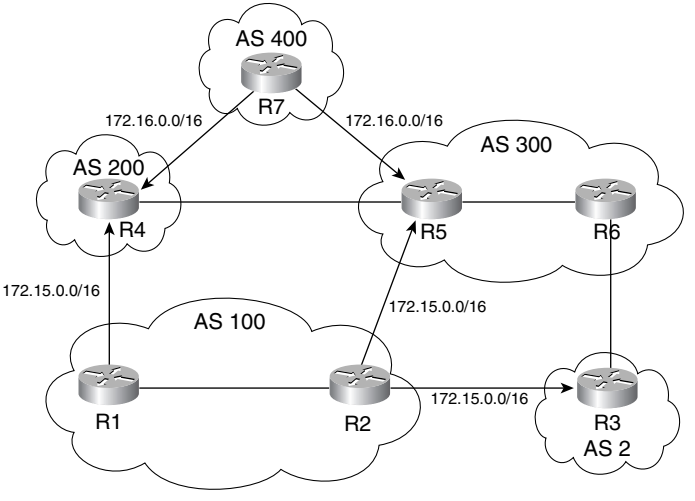
**Example 4-39** *Policy Accounting Statistics on FastEthernet10/0*

```
R3#show cef interface policy-statistics | begin FastEthernet10/0
FastEthernet10/0 is up (if_number 19)
  Corresponding hwidb fast_if_number 19
  Corresponding hwidb firstsw->if_number 19
BGP based Policy accounting is enabled
 Index         Packets            Bytes
     1          867256         86725600
     2               0                0
     3               0                0
     4               0                0
     5               0                0
     6               0                0
     7               0                0
     8               0                0
```

# Case Study: AS Integration via the Local AS

This case study shows you how to integrate two existing autonomous systems (AS 100 and AS 2) into one AS (AS 2) using the Local AS feature. A simple topology is shown in Figure 4-11. AS 100 is multihomed to three different autonomous systems: 200, 300, and 2. The prefix 172.15.0.0/16 is generated and advertised to neighboring autonomous systems. AS 100 also receives the prefix 172.16.0.0/16 generated by AS 400.

**Figure 4-11** *Network Topology for the Case Study*



For the purposes of this case study, the last octet of an IP address indicates the router number. Basic BGP configurations for R1 and R2 are shown in Examples 4-40 and 4-41, respectively.

**Example 4-40** *BGP Configuration on R1*

```
router bgp 100
 no synchronization
 bgp log-neighbor-changes
 network 172.15.0.0
 neighbor 192.168.12.2 remote-as 100
 neighbor 192.168.14.4 remote-as 200
 no auto-summary
```

**Example 4-41** *BGP Configuration on R2*

```
router bgp 100
 no synchronization
 bgp log-neighbor-changes
 network 172.15.0.0
 neighbor 192.168.12.1 remote-as 100
 neighbor 192.168.23.3 remote-as 2
 neighbor 192.168.25.5 remote-as 300
 no auto-summary
```

Examples 4-42 and 4-43 show the BGP RIB.

**Example 4-42** *BGP RIB on R1*

```
R1#show ip bgp
BGP table version is 3, local router ID is 192.168.14.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.15.0.0       192.168.12.2             0    100      0 i
*>                  0.0.0.0                  0         32768 i
* i172.16.0.0       192.168.25.5                100      0 300 400 i
*>                  192.168.14.4                         0 200 400 i
```

**Example 4-43** *BGP RIB on R2*

```
R2#show ip bgp
BGP table version is 3, local router ID is 192.168.25.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
```

**Example 4-43** *BGP RIB on R2 (Continued)*

```
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.15.0.0       192.168.12.1             0    100      0 i
*>                  0.0.0.0                  0         32768 i
*  172.16.0.0       192.168.23.3                           0 2 300 400 i
* i                 192.168.14.4                 100       0 200 400 i
*>                  192.168.25.5                           0 300 400 i
```

Now AS 100 and AS 2 decide to merge into a single AS 2. All BGP speakers in AS 100 are to be migrated to AS 2. Because a common IGP must be used in the same AS, IGP must be migrated first (migrating the IGP is outside the scope of this book and thus isn't covered here). To reduce migration risk and the impact on the peers, migration is to take a gradual approach, with R2 being migrated first.

Local AS is configured on R2 on the session with R5. To maintain the current forwarding architecture, a higher WEIGHT is set on R2 to prefer the path from R5. The outbound AS_PATH is prepended twice on R3 toward R6 and once on R1 toward R4. The **no-prepend** option on R2 is needed so that R1 accepts the path via R5, because now there is an eBGP session between R1 and R2.

Examples 4-44, 4-45, and 4-46 show the configurations on R1, R2, and R3, respectively.

**Example 4-44** *BGP Configuration on R1*

```
router bgp 100
 network 172.15.0.0
 neighbor 192.168.12.2 remote-as 2
 neighbor 192.168.14.4 remote-as 200
 neighbor 192.168.14.4 route-map Path-200 out
!
route-map Path-200 permit 10
 set as-path prepend 100
```

**Example 4-45** *BGP Configuration on R2*

```
router bgp 2
 network 172.15.0.0
 neighbor 192.168.12.1 remote-as 100
 neighbor 192.168.23.3 remote-as 2
 neighbor 192.168.25.5 remote-as 300
 neighbor 192.168.25.5 local-as 100 no-prepend
 neighbor 192.168.25.5 weight 100
```

**Example 4-46** *BGP Configuration on R3*

```
router bgp 2
 neighbor 192.168.23.2 remote-as 2
 neighbor 192.168.36.6 remote-as 300
 neighbor 192.168.36.6 route-map Path-300 out
!
route-map Path-300 permit 10
 set as-path prepend 2 2
```

The new BGP RIB on R1, R2, and R7 is shown in Examples 4-47, 4-48, and 4-49, respectively.

**Example 4-47** *BGP RIB on R1*

```
R1#show ip bgp
BGP table version is 3, local router ID is 192.168.14.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.15.0.0       192.168.12.2             0             0 2 i
*>                  0.0.0.0                  0         32768 i
*  172.16.0.0       192.168.12.2                           0 2 300 400 i
*>                  192.168.14.4                           0 200 400 i
```

**Example 4-48** *BGP RIB on R2*

```
R2#show ip bgp
BGP table version is 5, local router ID is 192.168.25.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.15.0.0       192.168.12.1             0             0 100 i
*>                  0.0.0.0                  0         32768 i
*> 172.16.0.0       192.168.25.5                         100 300 400 i
*                   192.168.12.1                           0 100 200 400 i
* i                 192.168.36.6                 100       0 300 400 i
```

**Example 4-49** *BGP RIB on R7*

```
R7#show ip bgp
BGP table version is 4, local router ID is 192.168.57.7
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
```

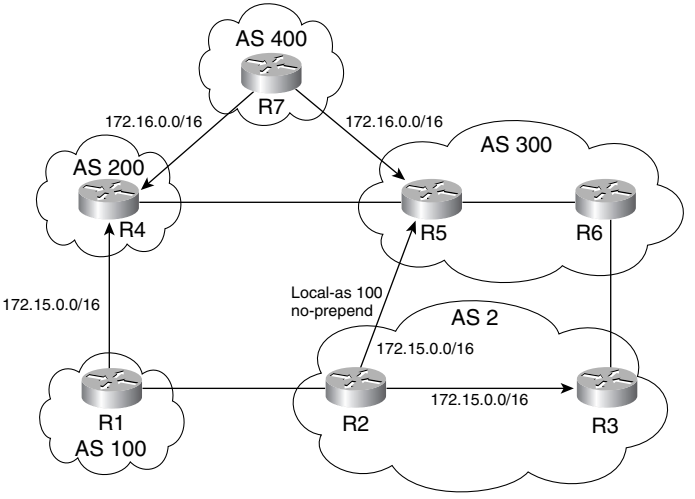
**Example 4-49** *BGP RIB on R7 (Continued)*

```
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.15.0.0       192.168.57.5                           0 300 100 2 i
*>                  192.168.47.4                           0 200 100 100 i
*> 172.16.0.0       0.0.0.0                  0         32768 i
```

The resulting topology is shown in Figure 4-12.

**Figure 4-12** *Topology After R2 Is Migrated to AS 2*




The next step is to migrate R1 to the new AS. Local AS is configured on R1 on the session with R4. AS_PATH prepending is now removed on R1. The LOCAL_PREF is modified to prefer the path via R4. The reason that LOCAL_PREF is used instead of WEIGHT is that R2 would also prefer the path via R1 for 172.16.0.0/16 if the link between R2 and R5 failed. The new BGP configurations on R1 and R2 are shown in Examples 4-50 and 4-51, respectively.

**Example 4-50** *BGP Configuration on R1*

```
router bgp 2
 network 172.15.0.0
 neighbor 192.168.12.2 remote-as 2
 neighbor 192.168.14.4 remote-as 200
 neighbor 192.168.14.4 local-as 100
```

*continues*

**Example 4-50** *BGP Configuration on R1*

```
 neighbor 192.168.14.4 route-map Set-lpref in
!
route-map Set-lpref permit 10
 set local-preference 120
```

**Example 4-51** *BGP Configuration on R2*

```
router bgp 2
 network 172.15.0.0
 neighbor 192.168.12.1 remote-as 2
 neighbor 192.168.23.3 remote-as 2
 neighbor 192.168.25.5 remote-as 300
 neighbor 192.168.25.5 local-as 100 no-prepend
 neighbor 192.168.25.5 weight 100
```

The BGP RIB is shown in Examples 4-52, 4-53, and 4-54 for R1, R2, and R7, respectively.

**Example 4-52** *BGP RIB on R1*

```
R1#show ip bgp
BGP table version is 3, local router ID is 192.168.14.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.15.0.0       192.168.12.2             0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 172.16.0.0       192.168.14.4                  120      0 100 200 400 i
* i                 192.168.25.5                  100      0 300 400 i
```

**Example 4-53** *BGP RIB on R2*

```
R2#show ip bgp
BGP table version is 5, local router ID is 192.168.25.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i172.15.0.0       192.168.12.1             0    100      0 i
*>                  0.0.0.0                  0         32768 i
* i172.16.0.0       192.168.14.4                  120      0 100 200 400 i
*>                  192.168.25.5                         100 300 400 i
* i                 192.168.36.6                  100      0 300 400 i
```

**Example 4-54** *BGP RIB on R7*

```
R7#show ip bgp
BGP table version is 5, local router ID is 192.168.57.7
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 172.15.0.0       192.168.57.5                           0 300 100 2 i
*                   192.168.47.4                           0 200 100 2 i
*> 172.16.0.0       0.0.0.0                  0         32768 i
```

Now AS 2 can convince AS 300 to change its peering and, thus, R5's configuration. Local AS is not needed on R2. However, AS 200 will only honor its previous peering agreement with AS 100. Local AS is still needed between R1 and R4. To maintain the same forwarding policy, R2 now needs to prepend its AS_PATH outbound to R5. The final configuration of R2 is shown in Example 4-55. The BGP RIB on R7 is shown in Example 4-56.

**Example 4-55** *BGP Configuration on R2*

```
router bgp 2
 network 172.15.0.0
 neighbor 192.168.12.1 remote-as 2
 neighbor 192.168.23.3 remote-as 2
 neighbor 192.168.25.5 remote-as 300
 neighbor 192.168.25.5 weight 100
 neighbor 192.168.25.5 route-map Path-300 out
!
route-map Path-300 permit 10
 set as-path prepend 2
```

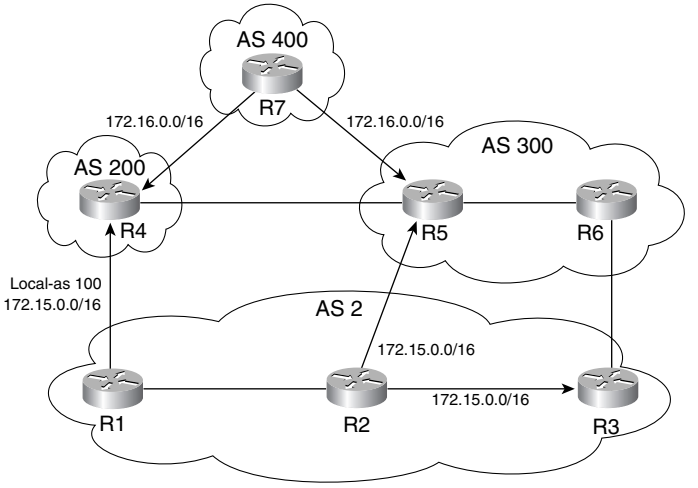
**Example 4-56** *BGP RIB on R7*

```
R7#show ip bgp
BGP table version is 10, local router ID is 192.168.57.7
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  172.15.0.0       192.168.47.4                           0 200 100 2 i
*>                  192.168.57.5                           0 300 2 2 i
*> 172.16.0.0       0.0.0.0                  0         32768 i
```

Figure 4-13 shows the final topology.

**Figure 4-13** *Final Topology*




# Summary

This chapter presented various techniques you can use to create complex and effective BGP policies. The chapter started with one of the fundamental techniques, regular expressions. Regular expressions are used extensively in IOS for pattern matching in parsing command outputs and in defining AS_PATH and community patterns.

A variety of filtering tools also were discussed. They include prefix lists, community lists, AS_PATH lists, route maps, and policy lists, all of which are used extensively in creating BGP policies. Additionally, more-complex policy tools were presented, including conditional advertisement, aggregation, deaggregation, Local AS, QoS policy propagation, and policy accounting. The chapter ended with a case study on AS merging using the Local AS feature.