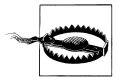


You have to be extra careful if the prefix you want to announce over BGP is already present in the global routing table. This happens in the situation where your ISP announced your address block but you now want to announce it yourself. Because a path to you, over your ISP, is always longer than a path to that ISP, your announcement will not be visible until your ISP stops announcing the same prefix. However, if something is misconfigured, the prefix is lost from the global routing table, and you are unreachable. This situation calls for close coordination with the ISP involved.

This applies only when you're going to announce the exact same address block/prefix your ISP was announcing and not a smaller (or bigger) one.

Note that the address blocks you announce over BGP don't have to match the way the address space is used internally. For instance, if the organization has been assigned the range `192.0.2.0/24`, but the ranges actually in use are `192.0.2.0/26` and `192.0.2.64/26`, the announcement over BGP should still be `192.0.2.0/24`.

Configuring the Router

We now have an AS number and address space, we know how we want to announce it, and we have a "numbered" network interface, so let's configure BGP:

1. The first step is to enter configuration mode and enable BGP:

   ```
   router bgp <AS number>
   ```

   The router should now show a *(config-router)#* instead of a *(config)#* prompt.

2. Add the address prefix to be announced:

   ```
   network <network address> mask <network mask>
   ```

   Note that the mask part doesn't show up in the configuration if the entered network/mask combination matches a valid classful net. In other words, *network 96.0.0.0 mask 255.0.0.0* simply shows as *network 96.0.0.0*.

3. Add the address and AS number of a neighboring router to the configuration:

   ```
   neighbor <ip address> remote-as <remote AS number>
   ```

4. Allow only routes that originate here to be announced to the neighboring AS:

   ```
   neighbor <ip address> filter-list 2 out
   ```

5. Just to be safe, also filter on IP address for outgoing updates:

   ```
   neighbor <ip address> distribute-list 3 out
   ```

6. Leave the BGP configuration mode and go back to regular configuration mode:

   ```
   exit
   ```

7. Add a static route that matches our BGP announcement. If there is no matching route in the routing table, the prefix won't be announced. Routing an address range to the Null0 interface makes sure BGP will always announce our prefix:

   ```
   ip route <network address> <network mask> null0 250
   ```


   250 is the administrative distance value. A high administrative distance ensures this static route doesn't get in the way of routes learned from routing protocols.

8. After this comes the outgoing AS path filter. The local AS number is added to the path *after* processing outbound AS path access lists, so we have to check for an empty path if we want to announce only our own locally sourced routes. An AS path filter list that allows only an empty AS path is a simple regular expression that matches the beginning of the line (^), immediately followed by the end of the line ($). Everything that isn't explicitly permitted is implicitly denied at the end of the access list, as with all Cisco access lists.

   ```
   ip as-path access-list 2 permit ^$
   ```

9. The address filter is a regular IP access list:

   ```
   access-list 3 permit host <network address>
   ```

Example 5-3 lists all the previously discussed changes necessary to create a full BGP configuration. It should be merged with the configurations from Appendix A and Example 5-2 to create a working configuration.

Example 5-3. Configuration changes to enable BGP

```
!
router bgp 60055
 network 192.0.2.0 mask 255.255.255.0
 neighbor 192.0.254.17 remote-as 40077
 neighbor 192.0.254.17 description BGP session to ISP A
 neighbor 192.0.254.17 filter-list 2 out
 neighbor 192.0.254.17 distribute-list 3 out
!
ip route 192.0.2.0 255.255.255.0 null0 250
!
ip as-path access-list 2 permit ^$
!
access-list 3 permit host 192.0.2.0
!
```

Monitoring BGP

If the other side has also configured BGP, a session should come up shortly after entering the BGP configuration commands. The *show ip bgp summary* command makes the Cisco router show the current BGP statistics, including for each neighbor a line that shows its current status, as shown in Example 5-4.

Example 5-4. Partial output of the show ip bgp summary command

```
BR1#show ip bgp summary
Neighbor        V    AS   [...]  Up/Down  State/PfxRcd
192.0.254.17    4 40077   [...]  never    Active
```

The last part of this line, "State/PfxRcd", is what concerns us here. If this shows a number, the BGP session is active, and the number indicates the number of routes received from the neighboring router. If it says `Idle`, there's a problem; probably the interface used to reach the neighbor is down. If the status is `Active`, the router is trying to initiate a BGP session, but the other side has not (yet) responded. `OpenSent` or `OpenConfirm` means the connection is being initialized.

Once the BGP session is established, the output of *show ip route* should list many routes starting with a `B`, indicating they have been learned from BGP:

```
BR1#show ip route
B    192.1.0.0/16 [20/0] via 192.0.254.17, 1w2d
B    192.5.4.0/23 [20/0] via 192.0.254.17, 1w2d
B    192.18.224.0/20 [20/0] via 192.0.254.17, 1w2d
```

Closer inspection of one of those routes doesn't show much BGP-specific information. Example 5-5 shows the route to the `F.ROOT-SERVERS.NET` root name server.

Example 5-5. A BGP-learned route in the routing table

```
BR1#show ip route 192.5.5.241
Routing entry for 192.5.4.0/23, supernet
  Known via "bgp 60055", distance 20, metric 0
  Tag 40077, type external
  Last update from 192.0.254.17 1w2d ago
  Routing Descriptor Blocks:
  * 192.0.254.17, from 192.0.254.17, 1w2d ago
      Route metric is 0, traffic share count is 1
      AS Hops 3
```

The BGP table, however, shows a wealth of BGP information. Example 5-6 uses the *show ip bgp* command to show BGP information for the route from Example 5-5.

Example 5-6. An entry in the BGP table

```
BR1#show ip bgp 192.5.5.241
BGP routing table entry for 192.5.4.0/23, version 3116521
Paths: (1 available, best #1)
  Not advertised to any peer
  40077 30099 3557
    192.0.254.17 from 192.0.254.17 (192.0.251.83)
      Origin IGP, localpref 100, valid, external, best, ref 2
```

Because we have configured only a single BGP peer so far, one of the most important differences between the routing table and the BGP table isn't visible: the BGP table stores all routes from all peers, but the routing table stores only one copy of each route for a certain destination prefix: the one the BGP route selection algorithm considers to be best.*

* Exceptions to this are possible when BGP is configured for load balancing.

Looking Glasses

Looking at your router's BGP and routing tables can help you determine if everything is working properly at your end, but it doesn't show whether your announcements are visible elsewhere on the Internet. Fortunately, many networks are kind enough to show you excerpts of their BGP table over the World Wide Web through a "looking glass." There are many looking glasses on the Net. They are easily found by searching for "bgp looking glass" using your favorite search engine. Digex (now Allegiance Business Internet) has four looking glasses and the looking-glass source code available at *http://nitrous.digex.net*.

Clearing BGP Sessions

Traditionally, after making changes to a BGP configuration that affect how routes to or from a neighbor are processed, the BGP session to that neighbor has to be cleared so that it's reinitialized. For instance, after updating an inbound AS path filter, clearing the session makes the remote router send all its routes so that the new filter can be applied. This is accomplished by issuing the *clear ip bgp* command followed by the IP address or AS number of the neighbor. However, resetting BGP sessions is disruptive; it causes all existing routes received from the BGP neighbor in question to be removed from the BGP and routing tables. Then there is a period of time during which there is no connectivity, or packets are diverted over an alternate path. When the routers decide to reestablish the session, the BGP and routing tables are repopulated, and connectivity is restored. (That is, if the entire process didn't trigger flap-dampening further upstream.)

Soft Reconfiguration Inbound

Having to go through all this just because of a minor routing policy change isn't much fun, so to avoid performing the full session-reestablishment procedure, Cisco implemented a soft reconfiguration option. When enabled for a peer, copies of all routes received from that peer are stored separately from the regular BGP table. After configuring a policy change, it's then possible to apply the new policy to the stored copies of the BGP information without having to reset the session. Soft reconfiguration is enabled as follows:

```
!
router bgp 60055
 neighbor 192.0.254.17 remote-as 40077
 neighbor 192.0.254.17 soft-reconfiguration inbound
!
```

A soft reconfiguration is then executed using the *clear ip bgp ... in* command. This works well, but it has a major disadvantage: it takes a lot of memory.

Prefix Lists

Prefix lists accomplish the same thing as distribute lists but in an easier to understand and more BGP-compatible manner. This prefix list allows only /20 and shorter prefixes in Class A and Class B space, and /24 and shorter prefixes in Class C space:

```
!
router bgp 60055
 neighbor 192.0.254.17 prefix-list infilter in
!
ip prefix-list infilter description inbound filter
ip prefix-list infilter seq 5 permit 0.0.0.0/1 le 20
ip prefix-list infilter seq 10 permit 128.0.0.0/2 le 20
ip prefix-list infilter seq 15 permit 192.0.0.0/3 le 24
!
```

The *le* keyword matches prefixes that are equal or shorter (less bits in the prefix, bigger blocks of address space). It's also possible to match equal or longer prefixes (more bits in the prefix, smaller blocks of address space) with the *ge* keyword or to specify a range by combining *ge* and *le*. The sequence numbers make it possible to delete individual lines or insert lines. You can't use distribute and filter lists at the same time for inbound or outbound filtering for a neighbor, but either can be combined with an AS path filter.

Internal BGP

Creating additional BGP sessions is straightforward: just adding the right *neighbor* statements to the BGP part of the configuration is enough. Don't forget to type *router bgp <as number>* to enter BGP configuration mode. As long as all BGP peers have similar properties (e.g., two peers are both ISPs), it's usually best to keep the filters for all BGP sessions the same. This keeps the number of access lists in the configuration limited to something manageable.

Having two connections to the Internet over different ISPs helps a lot to keep your network connected to the outside world in the presence of different kinds of external outages. But this means, the local network is now the weakest link. By removing single points of failure from the local network in addition to having multiple external links, you can at least make sure that any kind of failure won't affect connectivity to the *entire* network. This means a second BGP router, so two routers speak BGP with external ASes—and with each other. Adding another router for the second external connection is easily accomplished: you can pretty much copy the configuration of the first router and just change the Ethernet IP address and ISP-specific settings. Example 5-7 shows the BR2 configuration, with the security settings and access lists left out (these are the same as in the BR1 configuration).

- Routes over the main connection are preferred over routes that use a slower backup connection.
- Routes from peering connections are preferred over routes from transit connections.
- Routes directly to customers are preferred over external routes

Example 6-2 shows part of a BGP configuration where the routes received from both peers receive different Local Preference values.

Example 6-2. Setting the Local Preference for all routes received from a BGP neighbor

```
!
router bgp 60055
 neighbor 192.0.254.17 remote-as 40077
 neighbor 192.0.254.17 route-map ispa-in in
 neighbor 219.2.19.1 remote-as 50066
 neighbor 219.2.19.1 route-map ispb-in in
!
route-map ispa-in permit 10
 set local-preference 90
!
route-map ispb-in permit 10
 set local-preference 110
!
```

The *permit* keyword in the *route-map* statement means matched routes will be permitted to enter the BGP table or be propagated to the neighbor; a *deny* route map will filter out all routes matching the *match* clause. The number 10 is the sequence number, used to apply the different route maps with the same tag in the right sequence. In this case, there is only one route map for each tag (ispa-in and ispb-in), so the sequence number doesn't do anything.

Since we want to match all routes, there is no need to supply a *match* clause for the route maps. Both route maps just use a *set* clause to set the Local Preference for every route that is received from the respective neighbor. This has the effect that if ISP B has a route to a destination, this route will always be preferred over the route ISP A has to the same destination. Routes from ISP A will be used only if there is no matching route over ISP B. This would be a good routing policy if traffic over ISP B is a lot cheaper than traffic over ISP A. Example 6-3 shows the BGP table after applying the route maps.

Example 6-3. Partial BGP table with different Local Preferences

```
BR1#show ip bgp
BGP table version is 619734, local router ID is  192.0.254.18
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
   Network     Next Hop      Metric LocPrf Path
```

often. A more likely source of such attacks is an Internet Exchange. In many cases, it's possible for anyone connected to an IX switch to take over any IP and/or MAC address. Obviously, when someone takes over your peer's IP address, this will soon be noticed, but the damage may have already been done. For more information on connecting to an IX, see Chapter 12.

Even if an attacker can't take over a suitable IP address to make a BGP connection to your router, he can still use a second method to disrupt your network: by getting you or your peer to reset the BGP session. TCP incorporates a RST feature to reset sessions when one end of the connection no longer recognizes them. This usually happens when one system reboots during a TCP session. The other system continues to send packets until the first system is back online again. When this system receives a packet belonging to a TCP session it doesn't recognize, it sends back a RST packet to terminate the session. An attacker can falsify a RST packet to disconnect the BGP session between two routers. For private interconnects, this isn't such a big problem, because the falsified source addresses are easily filtered out. But a router can't tell if a packet coming in from the IX switch really came from the peer indicated by the source address or if it came from some other system that is also connected to the IX. The attacker would also have to know a valid 32-bit TCP sequence number, so this attack still isn't easy. An attacker who has intimate knowledge of the hardware and software involved, however, may be able to guess the sequence number within much fewer than the full 2^{32} tries.

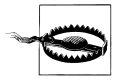
The solution to prevent these attacks from succeeding is to use a password for BGP sessions. Since the TCP RST attack happens at the TCP level, BGP authentication also operates at the TCP level. Every TCP packet that is part of a password-protected BGP session carries a TCP option containing an MD5 hash value over the TCP payload and the password. When a router receives such a packet, it recomputes the MD5 hash and checks whether it matches the one in the TCP option. If it does, the TCP packet is allowed through for further processing. If not, the packet is ignored. This way, only someone in possession of the password can create packets that will be processed successfully. Then even an attacker who can intercept and change packets at will can't inject false information in BGP. If it isn't possible to use passwords (or if you don't trust your peer 100%), it's a good idea to explicitly reject incoming routes to your own address space. Example 7-1 both filters and sets a password.

Example 7-1. Protecting BGP

```
!
router bgp 60055
 neighbor 192.0.254.17 password abc def ghi
 neighbor 192.0.254.17 prefix-list infilter in
!
ip prefix-list infilter description inbound filter
ip prefix-list infilter seq 5 deny 192.0.2.0/24 le 32
ip prefix-list infilter seq 10 permit 0.0.0.0/0 le 32
!
```


The password is case-sensitive ("secret" is a different password from "Secret" or "SECRET"). The password can include spaces, and it may be up to 80 characters long. The only limitation is that it may not begin with a number. Make sure you have the password written down somewhere, because some versions of IOS immediately encrypt it. After entering *... password abc def ghi*, typing *show running-config* will reveal something like *... password 7 00051105445F0E004F264447*. However, this isn't true for all IOS versions. The number 7 immediately following *password* indicates the password has been encrypted, so the router knows not to encrypt it again. A number 0 indicates the password has not (yet) been encrypted. If the other side hasn't configured the password/MD5 protection option, or the MD5 doesn't match, you'll see `%TCP-6-BADAUTH` error messages on the router's console, the log buffer and other places the router is configured to send logging information to. Note that you may even see these messages for BGP sessions that aren't active because the *neighbor ... shutdown* command is in effect: the TCP MD5 option processing is done before the configured state of the neighbor is considered.

The prefix list in this example denies routes for `192.0.2.0/24` and any subsets (longer prefixes), including single IP addresses. The second line allows all other routes, from the default route `0.0.0.0/0` all the way down to every possible `/32`.



The router may also accept the syntax *permit 0/0 le 32* instead of *permit 0.0.0.0/0 le 32*, but it takes this to mean the entire IPv6 address space, rather than the entire IPv4 address space.

Alternatively, you may want to explicitly configure which prefixes to accept from a peer and reject everything else. Doing this manually requires a lot of maintenance, however, because this information changes often. Using a tool to generate filters from a Routing Registry is problematic because not every network registers everything properly. Also, larger networks may announce hundreds of prefixes, so the filters can become long.

Avoiding Black Holes

The default behavior of BGP is to announce all routes to all peers. So if someone forgets to configure the appropriate filters, excess routes leak out. Then, when peers start to send their traffic for these destinations, the incorrectly filtered network becomes overloaded, and the black hole is firmly in place. An even worse type of black hole is the one resulting from improperly redistributing BGP into an IGP and then the IGP into BGP, as discussed in Chapter 10. One way to avoid black holes is to apply inbound filters and allow only a predefined list of prefixes for every peer. This approach works well, and many networks use it. The downside is the amount of maintenance needed to keep the filters up to date. Many networks inform their peers about new address ranges only shortly before they are put into use, so you must be prepared to perform daily filter updates if you choose this approach. If this is

CHAPTER 11
Providing Transit Services

Being a multihomed ISP with BGP-speaking customers isn't very different from just being a multihomed network in most regards, but some things need extra attention. A regular multihomed network sends out just its own routes, which makes for easy outbound filtering: allow out the BGP announcements you're trying to send, and nothing else. When you're providing transit services, it gets a bit more complex. You can, of course, filter out any announcements that don't fit your idea of how your customers should announce their routes. That way, however, you don't leave any room for traffic engineering or BGP-based anti-DoS measures. A better solution is to accept all reasonable announcements. You should also have a solution for customers with two connections to your network. And expect customers to come to you with questions about IP multicast and IPv6.

The examples in this chapter all use AS 40077 as the local AS number with different customer and upstream ISP connections in each example.

Route Filters

Outbound route filters to upstream networks (peers and transit ISPs) are important to make sure you send only the routes you want to source and those of your customers to your peers and upstream ISPs. You need two types of outbound route filters: AS path and prefix filters. Having both prefix and AS path filters is redundant in theory, but in practice there are many ways in which faulty routes can escape one check, so they must be filtered out by the other. Example 11-1 shows both plus an inbound prefix list filter and how the filters are applied to an individual BGP session and a peer group.

Example 11-1. Filtering outbound routes to transit ISPs and peers

```
!
router bgp 40077
 network 192.0.224.0 mask 255.255.224.0
 network 223.15.0.0 mask 255.255.0.0
 neighbor nap peer-group
 neighbor nap prefix-list in in
 neighbor nap prefix-list out out
 neighbor nap filter-list 10 out
 neighbor 169.254.30.8 remote-as 30088
 neighbor 169.254.30.8 prefix-list in in
 neighbor 169.254.30.8 prefix-list out out
 neighbor 169.254.30.8 filter-list 10 out
!
ip route 192.0.224.0 255.255.224.0 Null0
ip route 223.15.0.0 255.255.0.0 Null0
!
ip as-path access-list 10 permit ^(40077_)*$
ip as-path access-list 10 permit ^(40077_)*(60055_)+$
!
ip prefix-list out description outbound route filter
ip prefix-list out seq 5 permit 192.0.224.0/19 le 22
ip prefix-list out seq 10 permit 223.15.0.0/16 le 19
ip prefix-list out seq 15 permit 192.0.2.0/24 le 27
ip prefix-list out seq 20 permit 220.37.0.0/20 le 23
!
ip prefix-list in description inbound route filter
ip prefix-list in seq 5 deny 192.0.224.0/19 le 32
ip prefix-list in seq 10 deny 223.15.0.0/16 le 32
ip prefix-list in seq 15 permit 0.0.0.0/0 le 24
!
```

In this example, there are four address ranges: a `/19` and a `/16` announced by the local AS (AS 40077), and a `/24` and a `/20` announced by a customer (AS 60055). There are *network* statements for the locally sourced `/19` and `/16` announcements, but not for the customer routes: they must announce these address blocks themselves. The `out` prefix list allows all four address ranges and routes for prefixes that are up to three bits longer. This way, some deaggregation is possible to counteract DoS attacks or black holes, but the blocks can't get too small (and thus their number too large) to avoid running into trouble with upstream maximum prefix settings. The inbound `in` prefix list filter makes sure the router doesn't listen to others announcing the network's own address space by filtering out all prefixes matching the locally sourced address blocks or subsets thereof. (External announcements of customer address ranges are allowed so that they can reach the customer when their connection to this network is down.) The last line of the `in` filter permits all prefixes that aren't longer than 24 bits. The first line of AS path access list 10 allows all AS paths consisting of zero or more times the sequence 40077_ (the local AS number and a space character). The second line allows all AS paths with the local AS number zero or more times followed by one or more (+) times the customer AS number. The * and + wildcard characters are made to work on multicharacter strings (atoms in *regexp* speak) by enclosing them in parentheses. The `^` (beginning of the line) and `$` (end of line) special characters make sure the regular expressions match only lines consisting entirely of the desired expression and not all lines that include the desired expression.