Module 3 – BGP route filtering and advanced features

Objective: Using the network configured in Module 2, use various configuration methods on BGP peerings to demonstrate neighbour filtering and more advanced IOS features.

Prerequisite: Module 2

Topology:

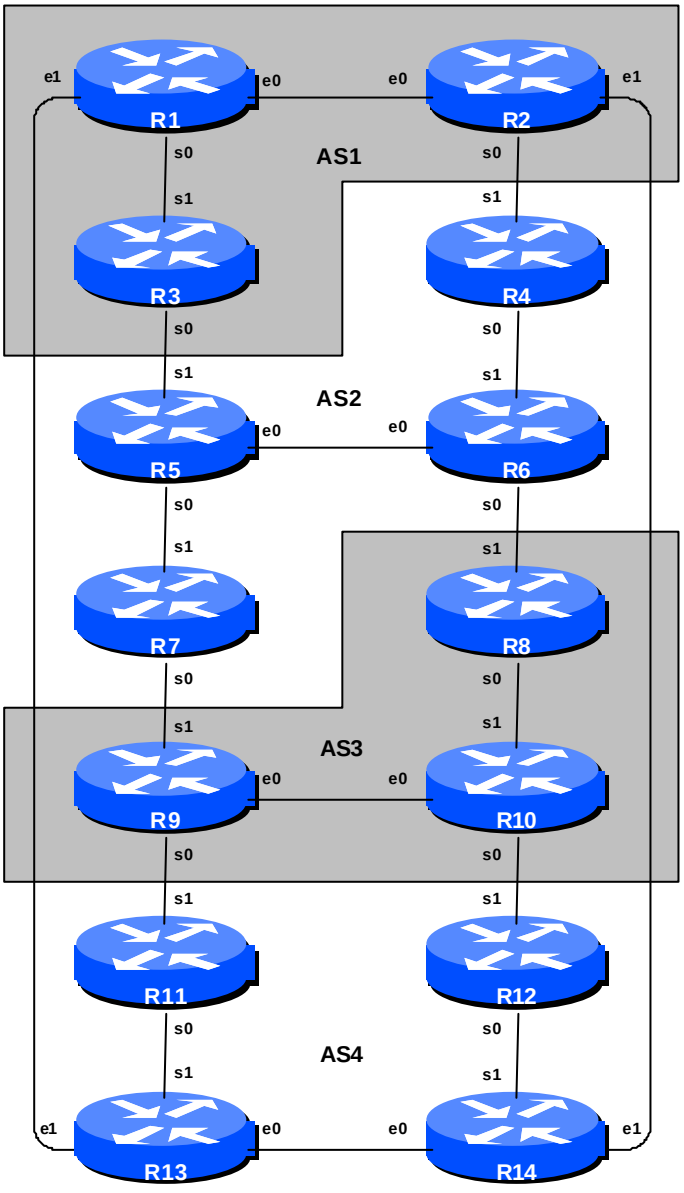


Figure 1 – BGP AS Numbers

Lab Notes

The previous Module provided an introduction to setting up external BGP, but provided no way of controlling which networks are announced to which AS. The purpose of this module is to introduce the student to the types of routing policy which are available using BGP.

In step 2) to 4) we want to configure each AS to become a **non-transit AS**, i.e. the AS won't allow a connecting AS's traffic to traverse it to reach another connecting AS. This will mean disconnectivity in the classroom – for example, AS3 will no longer be able to see any networks from AS1, etc. This is a deliberate policy to demonstrate the effectiveness of the BGP filtering being employed.

In steps 2) and 3) we achieve this by configuring an outgoing route filter to allow only local prefix(es) be sent to eBGP peers. At the same time, we also make sure that our peers only send us their own prefixes. We guarantee this by configuring an incoming filter. In general, it is good practice to configure filters in both directions to protect against misconfiguration at both ends of the peering. In step 4) we use AS-PATH filters. And in step 6) we make use of BGP communities to achieve the same effect.

Important: each step must be carried out and completed by the whole workshop **before** the next step can be started. Do not immediately start the next step without receiving the go-ahead from the workshop instructors. If you do, it is likely the routing will break, and other router teams may be unable to understand the results of the configuration they are trying to implement.

Important: retain the configuration used for Module 2, but remove the *aggregate-address* line from the BGP configuration.

Lab Exercise

1. **Implement BGP policies.** Before doing any configurations in this module it is important to note how to go about implementing BGP policies. Entering *prefix* lists, *as-path* filters or *route-map*s can be done at the CLI, but they only apply to BGP updates received after the policy configuration has been entered at the router. This is because BGP sends incremental updates describing changes in routes announced or withdrawn. To apply the policy to the entire BGP routing table received from or sent to the peer, the BGP session needs to be "reset". In older versions of IOS, this literally meant tearing down the BGP session, and then restoring it. However, as can be imagined, this causes severe stability issues on the service provider network, and 'RFC2918: Route Refresh Capability' has been added to most modern BGP implementations to allow graceful updates to BGP sessions when policy changes are required.

   To implement policy changes in any of the following worked examples, use the following router commands, for example to implement new policy inbound and outbound on the peering between Router 1 and Router 13:

```
Router1# clear ip bgp 100.1.2.2 out
Router1# clear ip bgp 100.1.2.2 in
```

Note: Do not forget the *in* and *out* subcommands in the above clear commands – omitting them will implement a hard reset of the BGP session. Review the BGP presentation if you don't understand why you do not want to ever do a hard reset on a BGP session.

Filtering using prefix-lists

2. **Configure prefix filter based on network address:** This step configures route prefix filtering based on network address. This is done using prefix-lists, and is one method of controlling networks which are exchanged in BGP peerings. The aim here is to configure eBGP peerings so that only networks from **neighbouring** ASes are exchanged.

Example: Router R13 (peering with R1)

```
!
ip prefix-list out-peer permit 100.4.0.0/18 le 32
ip prefix-list out-peer deny 0.0.0.0/0 le 32
!
ip prefix-list in-peer permit 100.1.0.0/18 le 32
ip prefix-list in-peer deny 0.0.0.0/0 le 32
!
router bgp 4
 no synchronization
 network 100.4.32.0 mask 255.255.240.0
 neighbor 100.1.2.1 remote-as 1
 neighbor 100.1.2.1 description eBGP peering with Router1
 neighbor 100.1.2.1 prefix-list out-peer out
 neighbor 100.1.2.1 prefix-list in-peer in
 no auto-summary
```

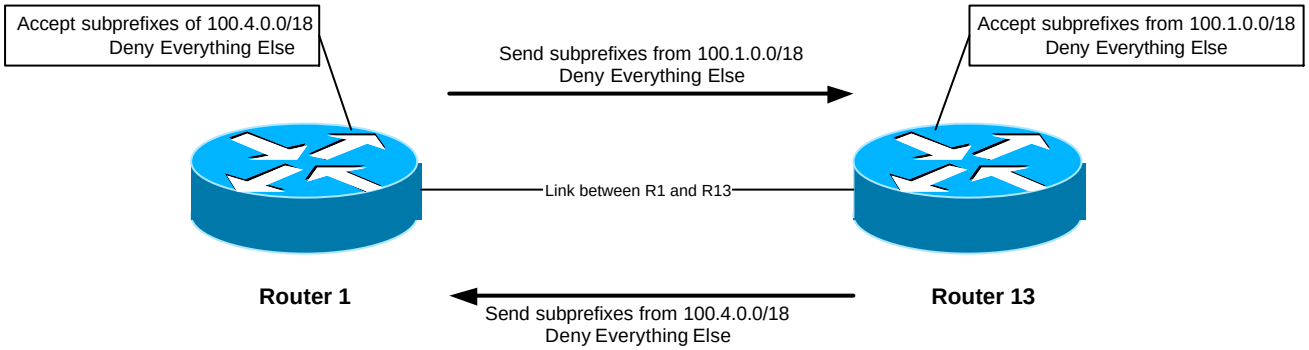
Example: Router R1 (peering with R13)

```
!
ip prefix-list out-peer permit 100.1.0.0/18 le 32
ip prefix-list out-peer deny 0.0.0.0/0 le 32
!
ip prefix-list in-peer permit 100.4.0.0/18 le 32
ip prefix-list in-peer deny 0.0.0.0/0 le 32
!
router bgp 4
 no synchronization
 network 100.1.0.0 mask 255.255.240.0
```

```
neighbor 100.1.2.2 remote-as 4
neighbor 100.1.2.2 description Peering with Router13
neighbor 100.1.2.2 prefix-list out-peer out
neighbor 100.1.2.2 prefix-list in-peer in
no auto-summary
```

Note: an IOS prefix-list always has an implicit *deny any* as the last statement even though it is not listed in the configuration. Some ISPs add the implicit *deny any* as they consider it good practice and a security precaution.

Note: these prefix-lists are only applied to peerings with other ASes. These are called **external** peerings (using eBGP). There is usually no need to apply such filters for iBGP peerings.



Checkpoint #1: *call the lab assistant to verify the connectivity. Each router team should check peerings to see the effect of this step. Use the "show ip bgp neigh x.x.x.x advertise|route" commands. Once complete and the lab instructors give you the go-ahead, remove the prefix-list configuration and access-lists, and carry on with step 2.*

STOP AND WAIT HERE

3. **Remove configuration from previous example.** This step will demonstrate how to remove the configuration entered in the previous example. This is essential before we move onto the next step.

Example: Router R1

```
Router1#conf t
Router1(config)#router bgp 1
!
! First remove prefix list from BGP peering with R13
!
Router1(config-router)#no neighbor 100.1.2.2 prefix-list out-peer out
Router1(config-router)#no neighbor 100.1.2.2 prefix-list in-peer in
```


Cisco Systems Inc
170 West Tasman Drive.
San Jose, CA 95134-1706
Phone: +1 408 526-4000
Fax: +1 408 536-4100

```
!
! Now remove the prefix-lists themselves
!
Router1(config)#no ip prefix-list out-peer
Router1(config)#no ip prefix-list in-peer
!
! That's the configuration nice and tidy, the way it should be.
!
Router1(config)#end
!
! Now clear the bgp peering so that the old policy is removed
!
Router1#clear ip bgp 100.1.2.2 out
Router1#clear ip bgp 100.1.2.2 in
Router1#
```

AS-PATH filters

4. **Configure prefix filter based on AS path attribute:** This step configures route prefix filtering based on AS path. This is done using AS path access-lists, and is another method of controlling networks which are exchanged in BGP peerings.

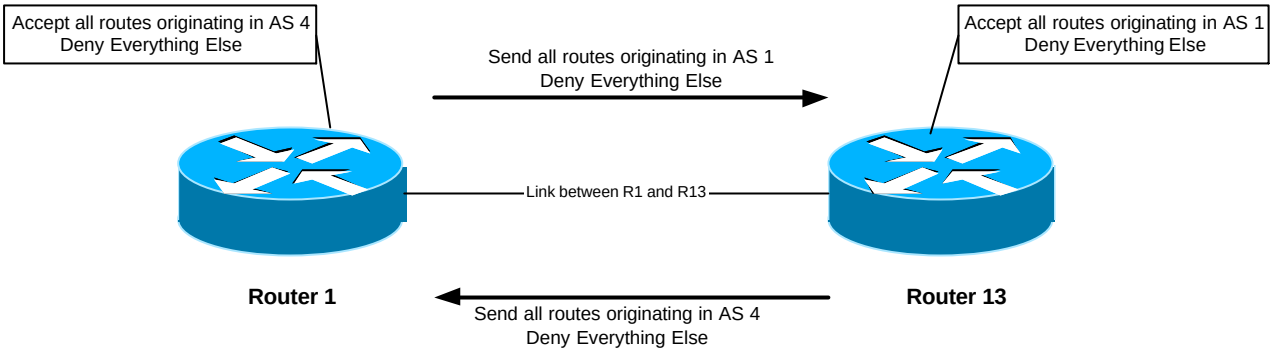
   Outgoing direction example – Router R13

```
ip as-path access-list 2 permit ^$
ip as-path access-list 3 permit ^1$
!
router bgp 4
 neighbor 100.1.2.1 remote-as 1
 neighbor 100.1.2.1 filter-list 2 out
 neighbor 100.1.2.1 filter-list 3 in
```

   Incoming direction example – Router R1

```
ip as-path access-list 2 permit ^$
ip as-path access-list 3 permit ^4$
!
router bgp 1
 neighbor 100.1.2.2 remote-as 4
 neighbor 100.1.2.2 filter-list 2 out
 neighbor 100.1.2.2 filter-list 3 in
```

   To verify that the regular expression works as intended, use the EXEC command *"**show ip bgp regexp** <regular-expression>"* to display all the paths that match the specified regular expression. Don't forget that the command *clear ip bgp <neighbour address>* is required to implement this filter.



Checkpoint #3: *call the lab assistant to verify the connectivity. Once the lab instructor gives the class the gohead, remove the attribute and filter list configuration and move on to the next step.*

STOP AND WAIT HERE

5. **Remove the configuration from the previous example.** This step will demonstrate how to remove the configuration entered in the previous example. This is essential before we move onto the next step.

   Example: Router 1

```
Router1#conf t
Router1(config)#router bgp 1
!
! First remove the filter list from BGP peering with R13
!
Router1(config-router)#no neighbor 100.1.2.2 filter-list 2 out
Router1(config-router)#no neighbor 100.1.2.2 filter-list 3 in
!
! Now remove the filter lists themselves
!
Router1(config)#no ip as-path access-list 2
Router1(config)#no ip as-path access-list 3
!
! That's the configuration nice and tidy, the way it should be.
!
Router1(config)#end
!
! Now clear the bgp peering so that the old policy is removed
!
Router1#clear ip bgp 100.1.2.2 in
Router1#clear ip bgp 100.1.2.2 out
Router1#
```


Cisco Systems Inc
170 West Tasman Drive.
San Jose, CA 95134-1706
Phone: +1 408 526-4000
Fax: +1 408 536-4100

BGP Communities for filtering (and route-maps)

6. **Introduction to BGP communities and route-maps:** This step introduces the Router Teams to using BGP communities for tagging, identifying, and eventually filtering prefixes. We achieve similar end results to what we achieved in the previous steps using prefix-list and as-path filters.

   On all routers, configure BGP to send a community for ***all*** prefixes belonging to the local AS advertised to external BGP peers. The community should be in the form of *[AS number]:[Router number]*. For example, Router9 should use community 3:9.

   Example on Router R9:

```
ip bgp-community new-format
! needed so that a community is treated in 16-bit:16-bit format
! rather than one 32-bit integer.
!
ip prefix-list out-match permit 100.3.0.0/18 le 32
!
route-map outfilter permit 10
 match ip address prefix-list out-match
 set community 3:9
!
router bgp 3
 neighbor x.x.x.x remote-as ASN
 neighbor x.x.x.x route-map outfilter out
 neighbor x.x.x.x send-community
```

   Note:
   1) A community attribute can be seen in the BGP table via *show ip bgp <network>* command.
   2) A community attribute is a 32-bit field. By IETF agreed-upon convention it is divided into two 16-bit fields for easy interpretation. The top 16-bit contains the AS number, while the lower 16-bit represents an integer that has specific meaning to the two ASes involved in the peering. The exceptions to this are the well-known community attribute strings such as ***no-export*** or ***local-as***.

   Q: Why is *send-community* needed for eBGP peerings?
   A: As community values are not passed by default between BGP peers, you need to tell the router explicitly to do this.

Checkpoint #4: *call the lab assistant to verify the connectivity. Each router team should again check their peerings to see what the effect is this time.*

STOP AND WAIT HERE

7. **Remove the configuration from the previous example.** This step will demonstrate how to remove the configuration entered in the previous example. This is essential before we move onto the next step.

   Example: Router 9

```
Router9#conf t
Router9(config)#router bgp 3
!
! First remove the route-map from the eBGP peering
!
Router9(config-router)#no neighbor x.x.x.x route-map outfilter out
!
! Now remove the route-map itself
!
Router9(config)#no route-map outfilter
!
! And now remove the prefix-list used by the route-map
!
Router9(config)#no ip prefix-list out-match
!
! That's the configuration nice and tidy, the way it should be.
!
Router9(config)#end
!
! Now clear the bgp peering so that the old policy is removed
!
Router9#clear ip bgp x.x.x.x out
Router9#
```

BGP Communities

8. **Setting BGP Communities.** In step 6 the community a network belongs to was generated at the point where one BGP router spoke to another BGP router. While this situation is valuable in demonstrating how to set communities, the more common scenario is where an ISP attaches a community to a network when the network is injected into the BGP routing table.

   Each router team should assign a community to the network block they have been allocated in Module 1. Review the BGP documentation to find out how to do this. Each router should set a community of format *[AS number]:[Router number]* exactly as in the previous step.

   Example for Router R1:

```
ip bgp-community new-format
!
route-map community-tag permit 10
 set community 1:1
!
router bgp 1
 no synchronization
 network 100.1.0.0 mask 255.255.240.0 route-map community-tag
 neighbor 100.1.2.2 remote-as 4
 neighbor 200.1.2.2 send-community
 no auto-summary
!
ip route 100.1.0.0 255.255.240.0 null0
```

Check that the network appears with its community in the BGP routing table.

Q: Why do your external, but not your internal, peers see the community set on the network?
A: See earlier. All peerings require the BGP *send-community* subcommand for the community attribute to be sent between BGP peers.

9. **Communities on internal BGP peerings.** Following on from the previous step, now set up the internal peerings so that the community attribute for your network is sent to local peers.

   Hint: to do this, simply add in the configuration line *neighbor x.x.x.x send-community* for all iBGP peerings. Don't forget to refresh the BGP peering sessions so that the configuration change can be implemented.

Checkpoint #5: *call the lab assistant and demonstrate how the community has been set for your network using the "show ip bgp" commands. Also, demonstrate that you can see the communities set by your internal and external BGP peers.*

10. **Configure incoming prefix filter based on community attribute.** The aim here is to only accept networks which are received from the neighbouring external BGP peer. (This is similar to what was being attempted in Steps 2 and 4 with prefix and AS path filtering.) For example, R13 should only accept the network originated by R1, and should use the knowledge of the community R1 has attached to the network to achieve this.

    Example on Router R13:

```
route-map infilter permit 10
 match community <community-list-no>
```

```
!
ip community-list <community-list-no> permit 1:1
!
router bgp 4
 neighbor 100.1.2.2 route-map infilter in
```

The <community-list-no> choice is up to each router team – it is not announced in any BGP peering or used in any other way apart from identifying the community-list (compare with the access-list number).

11. Set local-preference attribute on received eBGP routes. In this example, local preference will be set on the routes matching the community filter in Step 13. Retain the route-map used in Step 13 – an additional configuration line will be added to it. You also want to allow the other networks heard through the filters with the local-preference set to the default value.

Q. Why?
A. Without the second permit directive the route-map implements a default deny and no other prefixes will be passed through.

Example:

```
route-map infilter permit 10
 set local-preference 120
!
route-map infilter permit 20
```

Note that after a new policy is set, BGP session needs to be refreshed so that the new policy can then be applied. The router does not automatically keep all the updates it ever received from the peer so this is necessary. This is done by using "***clear ip bgp*** *<peer address>* ***in***" exec command.

Checkpoint #6: *call the lab assistant and demonstrate how the routes originated by your eBGP peers now have local preference set to 120. Also show how other routes have the default local preference of 100.*

Soft Reconfiguration

12. Soft-reconfiguration. Prior to Route Refresh (RFC2918) being standardised and implemented in IOS, Cisco had introduced a workaround for how to implement new BGP policies without doing a hard reset of the BGP session. This is called soft-reconfiguration, and while its use is more or less redundant[1] these

[1] More or less redundant in that some ISPs still find value in knowing exactly what prefixes their eBGP peer sent them as it helps with any troubleshooting. Route-refresh is still available on a peering that has soft-configuration configured.

days, it is still worthwhile exploring how it works, and what it does. When policy is implemented on a BGP session, prefixes which fail this policy are discarded by the router. However, with soft-reconfiguration enabled, the router will store the prefixes it would otherwise discard. If the policy is then changed, the router then has a complete list of the original prefixes sent to it by its peer, so can implement the new policy without having to tear down and re-establish the BGP session.

13. **Configure eBGP peer as soft-reconfiguration peer.** Set up soft-reconfiguration on your eBGP peering session. If you have two eBGP neighbours, set up soft-reconfiguration on one session only.

    Example on Router R1:

```
router bgp 1
 neighbor 100.1.2.2 soft-reconfiguration inbound
```

    Q. What happens to the eBGP peering?

    To implement policy changes, should you need to do so, the exec commands:

```
Router1#clear ip bgp 100.1.2.2 soft out
Router1#clear ip bgp 100.1.2.2 soft in
```

    will respectively reimplement the outbound and inbound policies on the eBGP peering. Omitting the *in* or *out* subcommand will apply the soft reset to the BGP session for both inbound and outbound policies.

14. Now go back to Step 11 and check the received prefix with "***show ip bgp*** *<address>*" where *<address>* is the prefix that we used route-maps to change attributes. Negotiate with your neighbouring AS so that they temporarily remove filters on the eBGP peering. Observe the output when you do "*show ip bgp <network>*". Notice the *received-only* comments in the output. What do you see? Write your answer here:

    With the soft-reconfiguration feature enabled, a ***show ip bgp*** *<n.n.n.n>* command will show all paths that have been received and whether they are accepted or not. A path that has been accepted with no change in any attribute (from an inbound filter) will be marked **received & used**. A path that has been denied or has had any attribute changed will be marked as **received-only**.

BGP Peer-Groups

15. **Configure the peer-group feature for iBGP peers.** BGP peer-groups help reduce the router processor load in sending updates to peers which have the same policy. This step configures BGP peer-groups for the iBGP peers in each AS. Replace the individual configuration for each iBGP peer with a peer-group configuration, as given in the example below.

    Example for Router R9:

```
router bgp 3
 neighbor ibgp-peers peer-group
 neighbor ibgp-peers description Peer-Group used for all iBGP peers
 neighbor ibgp-peers remote-as 3
 neighbor ibgp-peers update-source loopback 0
 neighbor ibgp-peers send-community
 neighbor 100.3.15.224 peer-group ibgp-peers
 neighbor 100.3.63.224 peer-group ibgp-peers
```

    Q: What are the advantages of using *peer-groups*?

    A: BGP peer groups allow a common configuration to be used for several BGP peers. The most common application is for iBGP. All internal BGP peers in an ISP network tend to have the same relationship with each other. Rather than having a substantial configuration per peer, and having to change every peering when details need to be changed, the configuration can be put in a peer-group, and only the peer group has to be changed to alter the peering configuration for all iBGP peers. This substantially reduces the work overhead required in making changes, the router CPU for processing, and significantly cleans up the configuration to view.

    It is strongly recommended that the *peer-group* subcommand be the "default" way of configuring all BGP peers. As was stated before, most iBGP peers have the same configuration, so it is of great benefit to the router, the operations staff, and network engineering staff to simplify the configurations wherever possible. Besides, a configuration which makes extensive use of peer-groups is usually much easier to read than one which has distinct configuration per peer, especially in networks with large numbers of peers.

    Note: Wherever BGP is configured in future in the workshop, it is expected that peer-groups will be used as part of the BGP configuration (especially for iBGP).

BGP Flap Damping

16. **Configure the BGP route-damping feature.** Consult the BGP presentation to remind yourself about the purpose of route-flap damping. It is another essential feature of the BGP configuration on a service provider's backbone routers. Flap damping tries to suppress the impact of the Internet routing table churn, giving more reliable service for customers, and more predictable network operation.

    Configuration Example:

```
router bgp 1
 bgp dampening
```

    Talk to other ASes and try triggering a network flap by removing the BGP network statement for their /20 network block. eBGP neighbours should see the prefix being withdrawn and attracting a penalty of 1000 due to the WITHDRAW. Now ask the neighbouring team to reinstall the BGP network statement. When it reappears in your BGP table, ask them to remove it again. Carry on doing this and after 3 flaps you will see that the prefix is now damped. Then check *show ip bgp flap* to see the penalty assigned to prefixes.

    After one flap you should see something like the following in the output of *sh ip bgp <prefix>* - this is taken from the lab of an ISP:

```
alpha#
BGP: charge penalty for 158.43.0.0/16 path 2830 with halflife-time 30
reuse/suppress 750/3000
       flapped 1 times since 00:00:00. New penalty is 1000
alpha#sh ip bgp 158.43.0.0
BGP routing table entry for 158.43.0.0/16, version 79
Paths: (1 available, no best path)
  Not advertised to any peer
  2830 (history entry)
    192.168.4.130 from 192.168.4.130 (192.168.9.13)
      Origin IGP, metric 0, localpref 100, external
      Dampinfo: penalty 992, flapped 1 times in 00:00:22

alpha#sh ip bgp 158.43.0.0
BGP routing table entry for 158.43.0.0/16, version 79
Paths: (1 available, no best path)
  Not advertised to any peer
  2830 (history entry)
    192.168.4.130 from 192.168.4.130 (192.168.9.13)
      Origin IGP, metric 0, localpref 100, external
      Dampinfo: penalty 984, flapped 1 times in 00:00:43
```

    Q. What does the history entry mean?

A. The history entry means that the eBGP neighbour has withdrawn the prefix and that there is a flap penalty attached to it. The prefix is not available to pass on to other BGP speakers, hence the *no best path* in the display. When the prefix is reannounced by the eBGP neighbour, the entry will change from *history* to being available again, as below:

```
alpha#sh ip bgp 158.43.0.0
BGP routing table entry for 158.43.0.0/16, version 80
Paths: (1 available, best #1)
  Advertised to non peer-group peers:
    192.168.12.1 192.168.18.2 192.168.20.1
  2830
    192.168.4.130 from 192.168.4.130 (192.168.9.13)
      Origin IGP, metric 0, localpref 100, valid, external, best
      Dampinfo: penalty 978, flapped 1 times in 00:01:02
alpha#
```

Cisco's default values are **bgp dampening 15 750 2000 60**. Each flap attracts a fixed penalty value of 1000.

The options for the *bgp dampening* command are:

15 – half-life (in minutes)
750 – reuse limit (penalty value at which the route will be reused)
2000 – suppress at (penalty value at which the route will be suppressed)
60 – suppress limit (max time in minutes the route will be suppressed)

Penalty decays at a granularity of 5 seconds. So every 5 seconds, the penalty will be reduced according to the half-life time (exponential decay). One flap attracts a penalty of 1000 units. So in this example, the route will be suppressed the first time it flaps. Once 800 units have been subtracted from the flap, the route will be reannounced. Once the penalty drops below half the reuse limit, the flap information is deleted from the flap table – the next flap will start counting from zero again.

17. Controlled BGP Damping:

Controlled BGP damping is more commonly implemented in the Internet today. For example, one common implementation is that as recommended by the RIPE Routing Working Group, where /24 prefixes attract different rates of damping compared with /22's and /23's, and the remaining prefix sizes.

Try using a route-map for more precise control of damping. This example damps the routes listed in prefix-list damp-prefix.

```
ip prefix-list damp-prefix permit x.x.x.x/m
```

```
!
route-map damp-some permit 10
 match ip address prefix-list damp-prefix
 set dampening 15 750 2000 60
!
router bgp 200
 bgp dampening route-map damp-some
```

Checkpoint #7: *call the lab assistant to verify the operation of damping and its configuration.*

18. Summary: This module has introduced some of the basic features available to configure BGP peerings in Cisco IOS. The reader is encouraged to try further permutations of the configuration examples given here. Community usage is gaining in popularity as the feature is now recognised to give considerable advantages in controlling routing policy between different ASes. BGP route flap damping, soft reconfiguration, and peer-groups are also widely used in ISP backbones as they considerably ease administration and configuration of an operational network. For recommended operational parameters for BGP route flap damping configuration for Internet connected sites, the reader should consult http://www.ripe.net/docs/ripe-229.html and the **Cisco ISP Essentials** book available from Cisco Press. For more information about the algorithms behind Cisco's implementation of route-flap damping, the reader should consult RFC2439.

Review Questions:

1. Why is making use of Route Refresh the best way of implementing new BGP policy?

2. Why do AS-PATH filters provide less granularity then prefix-filters for filtering a BGP session? And which is preferred in an ISP network, and why?

3. When should the community attribute be set on a BGP prefix?

4. Does IOS send BGP communities by default for iBGP? For eBGP? If not, what must operators remember to do?

5. Why would route-flap damping on an iBGP session be potentially harmful to a network's operation?

CONFIGURATION NOTES

Documentation is critical! You should record the configuration at each ***Checkpoint***, as well as the configuration at the end of the module.