IPv6 Module 7 – BGP Route Filtering and Advanced Features

Objective: Using the network configured in Module 6, use various configuration methods on BGP peerings to demonstrate neighbour filtering and more advanced IOS features.

Prerequisite: Module 6

Topology:

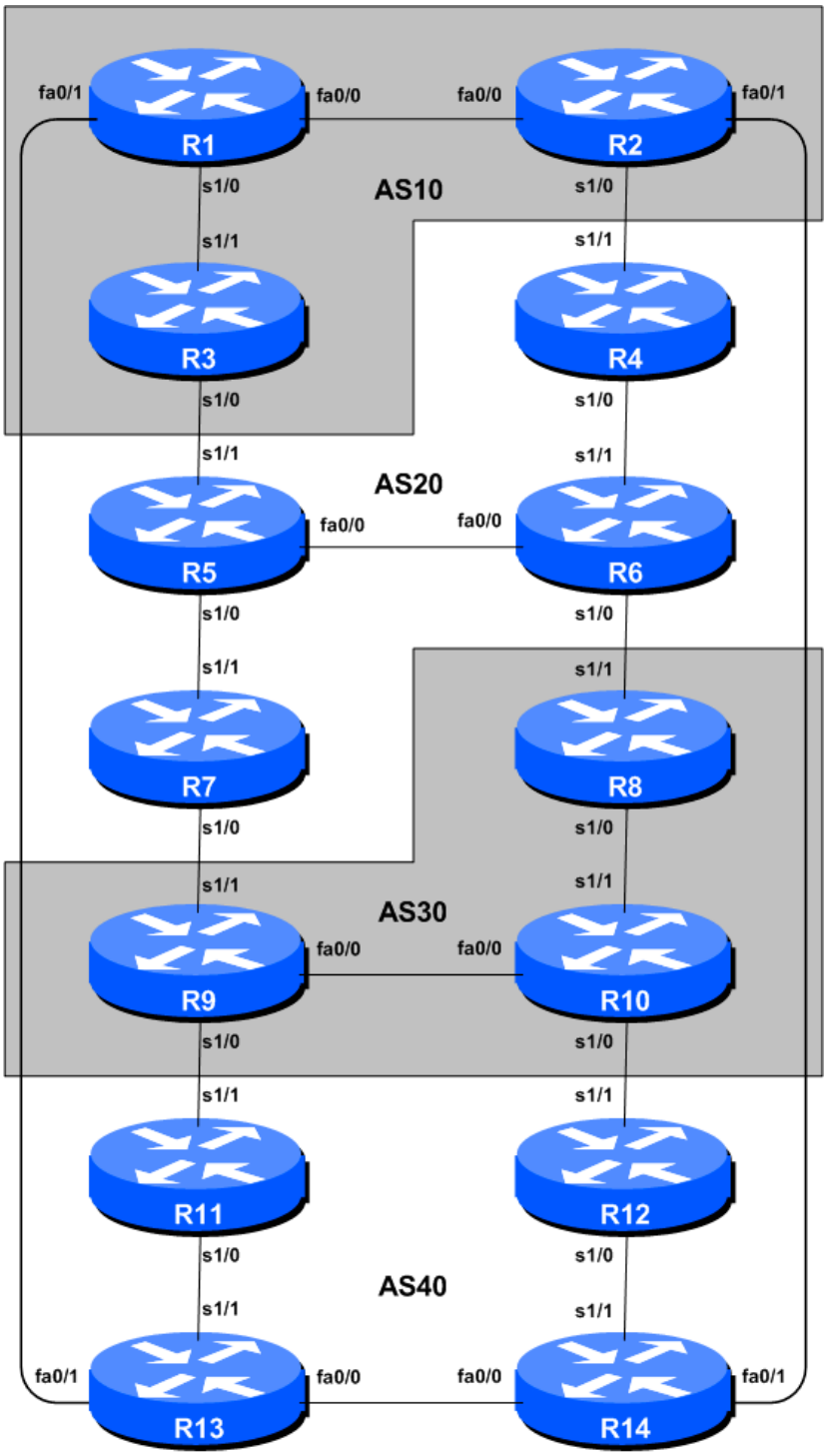


Figure 1 – BGP AS Numbers

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 steps 2 to 6 we will 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, AS30 will no longer be able to see any networks from AS10, etc. This is a deliberate policy to demonstrate the effectiveness of the BGP filtering being employed.

In step 2 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 the main part of Module 6.

Note: this IPv6 module can be completed independently of the IPv4 version of this module. It only requires the basic topology and connectivity provided by Module 6.

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 bgp ipv6 unicast 2001:db8:0:4::1 out
Router1# clear bgp ipv6 unicast 2001:db8:0:4::1 in
```

Note 1: 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.

Note 2: Rather than refreshing the BGP session using the neighbour IP address, it is also possible to refer to the neighbour by its ASN. Since Router 13 is in AS40, the alternative commands would be:

```
Router1# clear bgp ipv6 unicast 40 out
Router1# clear bgp ipv6 unicast 40 in
```

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 bgp ipv6 unicast" command to show the BGP table – ensure that the external prefixes learned by BGP now have a local next-hop address. Use the "trace" command to show that network connectivity is unaffected.*

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)

```
!
ipv6 prefix-list v6out-peer permit 2001:dbb::/32
ipv6 prefix-list v6out-peer deny ::/0 le 128
!
ipv6 prefix-list v6in-peer permit 2001:db8::/32
ipv6 prefix-list v6in-peer deny ::/0 le 128
!
router bgp 40
 address-family ipv6
  no synchronization
  network 2001:dbb::/32
  neighbor 2001:db8:0:4:: remote-as 10
  neighbor 2001:db8:0:4:: description eBGP peering with Router1
  neighbor 2001:db8:0:4:: prefix-list v6out-peer out
  neighbor 2001:db8:0:4:: prefix-list v6in-peer in
!
```

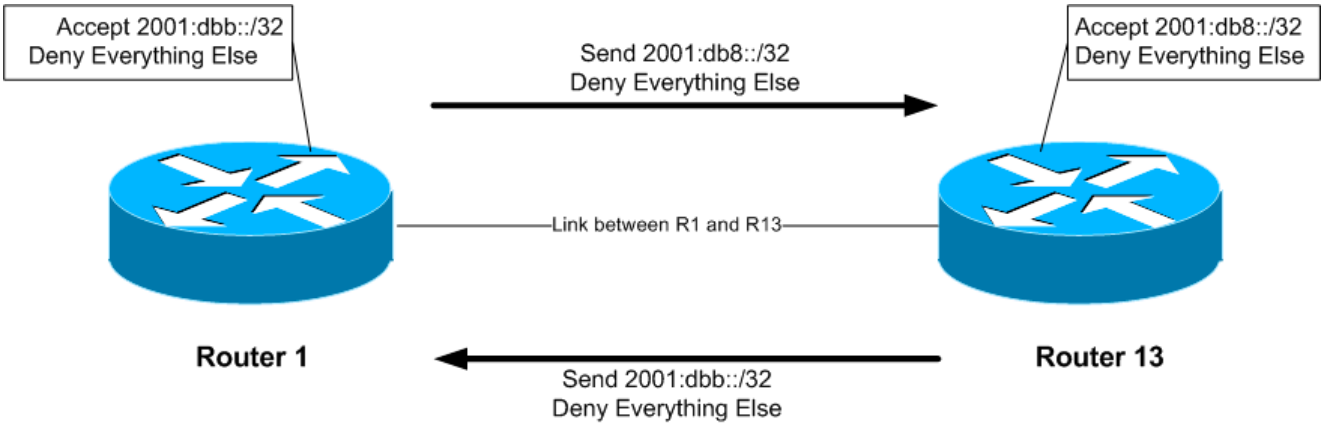
   Example: Router R1 (peering with R13)

```
!
ipv6 prefix-list v6out-peer permit 2001:db8::/32
ipv6 prefix-list v6out-peer deny ::/0 le 128
!
ipv6 prefix-list v6in-peer permit 2001:dbb::/32
ipv6 prefix-list v6in-peer deny ::/0 le 128
!
router bgp 10
 address-family ipv6
```

```
  no synchronization
  network 2001:db8::/32
  neighbor 2001:db8:0:4::1 remote-as 40
  neighbor 2001:db8:0:4::1 description Peering with Router13
  neighbor 2001:db8:0:4::1 prefix-list v6out-peer out
  neighbor 2001:db8:0:4::1 prefix-list v6in-peer in
!
```

Note: an IOS prefix-list always has an implicit *deny* everything as the last statement even though it is not listed in the configuration. Some ISPs add the implicit *deny* everything 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 #2: *call the lab assistant to verify the connectivity. Each router team should check peerings to see the effect of this step. Use the "show bgp ipv6 uni neigh x.x.x.x advertise|route" commands.*

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 10
Router1(config-router)#address-family ipv6
!
! First remove prefix list from BGP peering with R13
!
Router1(config-router-af)#no neighbor 2001:db8:0:4::1 prefix-list v6out-peer out
Router1(config-router-af)#no neighbor 2001:db8:0:4::1 prefix-list v6in-peer in
!
! Now remove the prefix-lists themselves
!
Router1(config)#no ipv6 prefix-list v6out-peer
Router1(config)#no ipv6 prefix-list v6in-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 bgp ipv6 unicast 2001:db8:0:4::1 out
Router1#clear bgp ipv6 unicast 2001:db8:0:4::1 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.

   Example: Router R13 (peering with R1)

```
ip as-path access-list 2 permit ^$
ip as-path access-list 3 permit ^10$
!
router bgp 40
 address-family ipv6
  neighbor 2001:db8:0:4:: remote-as 10
  neighbor 2001:db8:0:4:: filter-list 2 out
  neighbor 2001:db8:0:4:: filter-list 3 in
!
```

   Example: Router R1 (peering with R13)

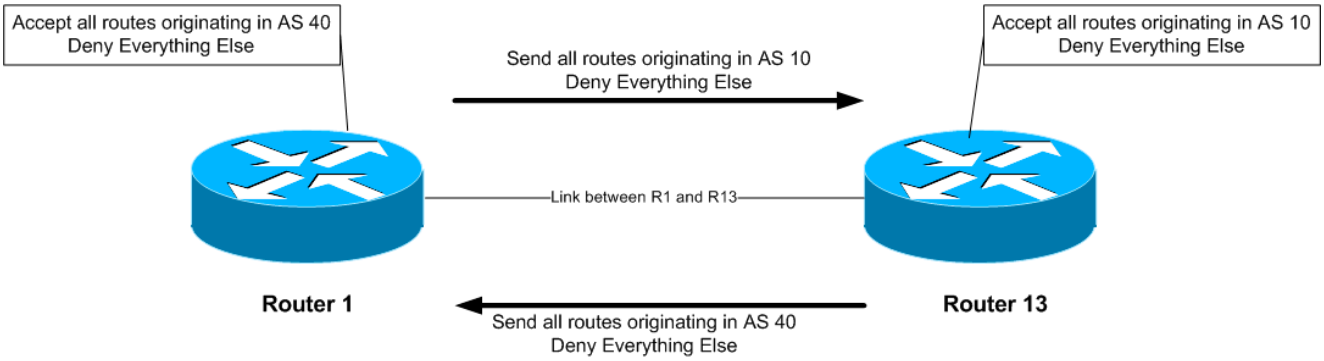
```
ip as-path access-list 2 permit ^$
ip as-path access-list 3 permit ^40$
!
router bgp 10
 address-family ipv6
  neighbor 2001:db8:0:4::1 remote-as 40
  neighbor 2001:db8:0:4::1 filter-list 2 out
  neighbor 2001:db8:0:4::1 filter-list 3 in
!
```

   Q. Why does the outbound filter list match the null as-path and not the local AS number in the above examples?

   A. Because the AS-PATH attribute is set after the prefix lists, as-path filters and route-maps. If the local AS was included in the outbound filter-list configuration, the prefixes would be ignored as their AS-PATH attribute would not be set at that stage.

   To verify that the regular expression works as intended, use the EXEC command *show bgp ipv6 unicast regexp <regular-expression>* to display all the paths that match the specified regular expression. Don't forget that the command *clear bgp ipv6 unicast <neighbour address> [in|out]* is required to implement this filter.

Notice that the AS Path filters allow all prefixes originated by the neighbouring AS – in the previous example using prefix filters, only the aggregate from the neighbouring AS got through the filters.



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 10
Router1(config-router)#address-family ipv6
!
! First remove the filter list from BGP peering with R13
!
Router1(config-router-af)#no neighbor 2001:db8:0:4::1 filter-list 2 out
Router1(config-router-af)#no neighbor 2001:db8:0:4::1 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 bgp ipv6 unicast 2001:db8:0:4::1 in
Router1#clear bgp ipv6 unicast 2001:db8:0:4::1 out
Router1#
```

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 30:9.

   Example on Router R8:

```
! Display communities in 16-bit:16-bit format rather than
! as one 32-bit integer.
!
ip bgp-community new-format
!
ipv6 prefix-list out-match permit 2001:dba::/32 le 48
!
route-map outfilter permit 10
 match ipv6 address prefix-list out-match
 set community 30:8
!
router bgp 30
 address-family ipv6
  neighbor 2001:dba:0:1::1 remote-as 20
  neighbor 2001:dba:0:1::1 route-map outfilter out
  neighbor 2001:dba:0:1::1 send-community
!
```

   Note:
   1) A community attribute can be seen in the BGP table via *show bgp ipv6 unicast <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 30
Router9(config-router)#address-family ipv6
!
! First remove the route-map from the eBGP peering
!
Router9(config-router-af)#no neighbor 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 ipv6 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 bgp ipv6 unicast 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 ***customer network block*** (the /48) they originate in BGP. 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 R3:

```
ip bgp-community new-format
!
route-map community-tag permit 10
 set community 10:3
!
router bgp 10
 address-family ipv6
  no synchronization
  network 2001:db8:3::/48 route-map community-tag
  neighbor 2001:db9:0:1:: remote-as 20
```

```
  neighbor 2001:db9:0:1:: send-community
!
ipv6 route 2001:db8:3::/48 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 bgp ipv6 unicast" 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:

```
ip community-list 5 permit 10:1
!
route-map infilter permit 10
 match community 5
!
router bgp 40
 address-family ipv6
  neighbor 2001:db8:0:4:: route-map infilter in
!
```

    The community-list number 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 10. Retain the route-map used in Step 10 – 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
 match community 5
 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 bgp ipv6 unicast*** <*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.*

BGP Peer-Groups

12. 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 30
 address-family ipv6
  neighbor v6-ibgp-peers peer-group
  neighbor v6-ibgp-peers description Peer-Group used for all iBGP peers
  neighbor v6-ibgp-peers remote-as 30
  neighbor v6-ibgp-peers update-source loopback 0
  neighbor v6-ibgp-peers send-community
  neighbor v6-ibgp-peers next-hop-self
  neighbor v6-ibgp-peers password cisco
!
```

Note: the old pre-*peer-group* configuration **must** be removed before converting from non-*peer-group* configuration to using peer-groups.

```
router bgp 30
 no neighbor 2001:dba::1
 no neighbor 2001:dba::3
!
 address-family ipv6
  neighbor 2001:dba::1 peer-group v6-ibgp-peers
  neighbor 2001:dba::1 activate
  neighbor 2001:dba::3 peer-group v6-ibgp-peers
  neighbor 2001:dba::3 activate
!
```

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 (and most definitely for any iBGP configuration).

13. **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. Route refresh and peer-groups are also widely used in ISP backbones as they considerably ease administration and configuration of an operational network.

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?