

AlliedWare Plus™ & AlliedWare OS

How To | Use Route Maps and Other Filters to Filter and Alter BGP and OSPF Routes

Introduction

ISPs transport large volumes of data. They often have to pay large amounts of money to transport their data through hired links, or through other providers' networks. Similarly, they can also charge money for transporting other ISPs' data through their network.

Where significant amounts of money are involved, there are typically complex negotiations involved, and agreements made that are bound by all sorts of rules and restrictions and guarantees.

Hence, ISPs need to be able to very precisely control which data gets sent and received on which links. This is achieved by having very precise control over the way the routing tables in their routers are built.

To that end, the BGP implementation on ATI routers and switches includes a set of facilities for filtering routes, and for altering the attributes that are associated with certain routes in BGP update messages. The main purpose of this document is to give an overview of these features, and explain how to configure them. One of the central route manipulation facilities is the **route map**.

Route maps can also be used for manipulating OSPF routes, so this document concludes by describing the use of route maps for OSPF.

Contents

Introduction	1
Contents	2
Which products and software versions does this How To Note apply to?	2
Related How To Notes	2
BGP: Concepts and terminology	3
BGP peers	3
BGP updates	3
Update attributes	4
BGP: Overview of the available filter types	5
Filter types	5
Hierarchy of the different filters	6
BGP: Example	6
Basic configuration	7
BGP: Configuring distribute filters	8
About ACLs	8
Using ACLs as filters	9
Example: Distribute filters	10
BGP: Configuring AS path filters	12
AS path lists	12
Using AS path lists as path filters	13
Example: AS path filters	13
BGP: Configuring prefix filters	16
About prefix lists	16
Mask length	17
Using prefix lists as prefix filters	17
Example: Prefix filters	18
BGP: Configuring route maps	20
Structure of a route map	20
Clauses	20
Configuring a match clause	20
Configuring a set clause	24
The effect of different combinations of clauses	26
BGP: Applying distribute, path, prefix, and route map filters to a peer	30
Examples	32
BGP: Applying route maps to imported routes	41
Syntax	41
Other uses of route maps	41
neighbor default-originate	41
neighbour unsuppress-map	42
network	42

show ip bgp	42
BGP: Route map filtering example	43
BGP configuration	43
Route map configuration	43
OSPF: Configuring route maps for filtering and modifying OSPF routes	44
Configuring a match clause	45
Configuring a set clause	46
OSPF: Applying route maps	47

Which products and software versions does this How To Note apply to?

This configuration applies to **AlliedWare Plus** software version 5.2.2-0.4 and above, for the following Allied Telesis switches:

- SwitchBlade x908
- x900-12XT/S
- x900-24 series
- x600 Series

Note: This document is a revision of an AlliedWare document, and also applies to the following AlliedWare products and software versions.

AlliedWare Products: AR440S, AR441S, AR450S, AR725, AR745, AR750S, Rapier i series, AT-8800 series, AT-9800 series, AT-8948, AT-9924T, AT-9924SP, AT-9924T/4SP, AT-9924Ts, x900 series.

AlliedWare Software version: 2.7.4 and above.

Related How To Notes

You can also configure route maps on routers and switches running AlliedWare. For details, see the AlliedWare Note *How To Use Route Maps and Other Filters to Filter and Alter BGP and OSPF Routes*. This Note is available from www.alliedtelesis.com/resources/literature/howto.aspx.

BGP: Concepts and terminology

Before moving on to look at the filtering processes, it is important to first have some understanding of certain aspects of how BGP works. The following sections describe:

- BGP peers
- BGP updates
- Update attributes

BGP peers

Definition Within the BGP protocol, the exchange of routing information is carried out between pairs of routers. Two routers create a TCP connection with each other, and exchange routing information as specific data packets within that TCP session. The routers at the ends of the TCP connection are referred to as BGP peers. Any given router can form peering relationships with multiple routers.

Usually a BGP router with an ISP will form peer relationships with BGP routers at other ISPs or clients with which it has entered into data transporting agreements.

The process of BGP filtering usually comes down to a matter of specifying the routes that will be sent to, or received from, each of a router's peers.

BGP updates

Definition Once a router has established a BGP connection with a peer, it will start to exchange routing information with that peer. A BGP update message is the packet that is used to transfer the routing information.

The routing information contained within an update message consists of:

- a set of attribute values (see the next section for a description of the possible attributes)

and

- a list of one or more prefixes. A prefix is the network portion of an IP address, in dotted decimal notation, optionally followed by a "/" character and a decimal number from 0 to 32. Each prefix contained within an update message represents a network that can be reached through the IP address given in the NextHop attribute contained in the same update message.

Note: There is only one NextHop attribute in an update message, so all the routes in the update message have the same next hop.

Update attributes

As mentioned above, each BGP update message contains a set of attributes. These attributes describe some of the properties of the routes, and can be used in making decisions about which routes to accept and which to reject.

Some of the attributes are:

Origin

How a prefix came to be routed by BGP at the origin AS. Prefixes are learned from various sources such as directly connected interfaces, manually configured static routes, or dynamic internal or external routing protocols, and then put into BGP.

AS-path

The list of ASes through which the announcement for the prefix has passed. As prefixes pass between ASes each AS adds its Autonomous System Number (ASN).

Next-hop

The address of the next node that the router should send packets destined for the specified prefixes to, in order to get the packets closer to the destination.

Multi-Exit-Discriminator (MED)

A metric expressing the optimal path to reach a particular prefix in or behind a particular AS.

Local-preference

A metric used in IBGP so each host knows which path inside the AS it should use to reach the advertised prefix. EBGP peers do not send this value, and ignore it on receipt.

Atomic-aggregate

A non-transitive attribute that allows BGP peers to inform each other about decisions they have made regarding overlapping routes. Non-transitive means that if the attribute is received by a device that does not recognise the attribute, it is dropped and not passed on to the next router.

Aggregator

Can be attached to an aggregated prefix to specify the AS and router that performed the aggregation.

Community

Indicates where a prefix is relevant to—for example, if it is relevant to the whole Internet, or just within an AS.

BGP: Overview of the available filter types

The following sections describe the various types of filters that can be applied to BGP updates and the hierarchy of the filters.

Filter types

There are four filter types that can be applied to the BGP updates being exchanged between BGP peers:

Distribute filters

These use ACLs and look at the individual prefixes within an update message. If a prefix within the update message matches the filter criteria then that individual prefix is filtered out or accepted depending on what action the filter entry has been configured to carry out. Note that you cannot combine distribute filters and prefix filters.

Path filters

These look at the AS-Path attribute in update messages. If the AS-Path attribute in the update matches the filter criteria then the whole update message is filtered out or accepted, depending on what action the filter entry has been configured to carry out.

Prefix filters

These use prefix lists and look at the individual prefixes within an update message. If a prefix within the update message matches the filter criteria then that individual prefix is filtered out or accepted depending on what action the filter entry has been configured to carry out. Note that you cannot combine distribute filters and prefix filters.

Route maps

These have a complex combination of match criteria and actions. They can be used to filter out routes and also to alter the attributes in update messages.

Note: All these filter types can be used in incoming or outgoing directions. So, all the filters can all be used to filter the update packets that are received from a peer, or the update packets which the router itself is sending to a peer.

Hierarchy of the different filters

For distribute filters (ACLs), path filters, and prefix filters, the order of application is not important. If an update is denied by any given filter, it is discarded immediately, and is not run through any of the other filters. If an update is permitted by one filter, it is passed through to the next filter to be considered. At the end, you end up with the set of updates that all the filters agree should not be discarded.

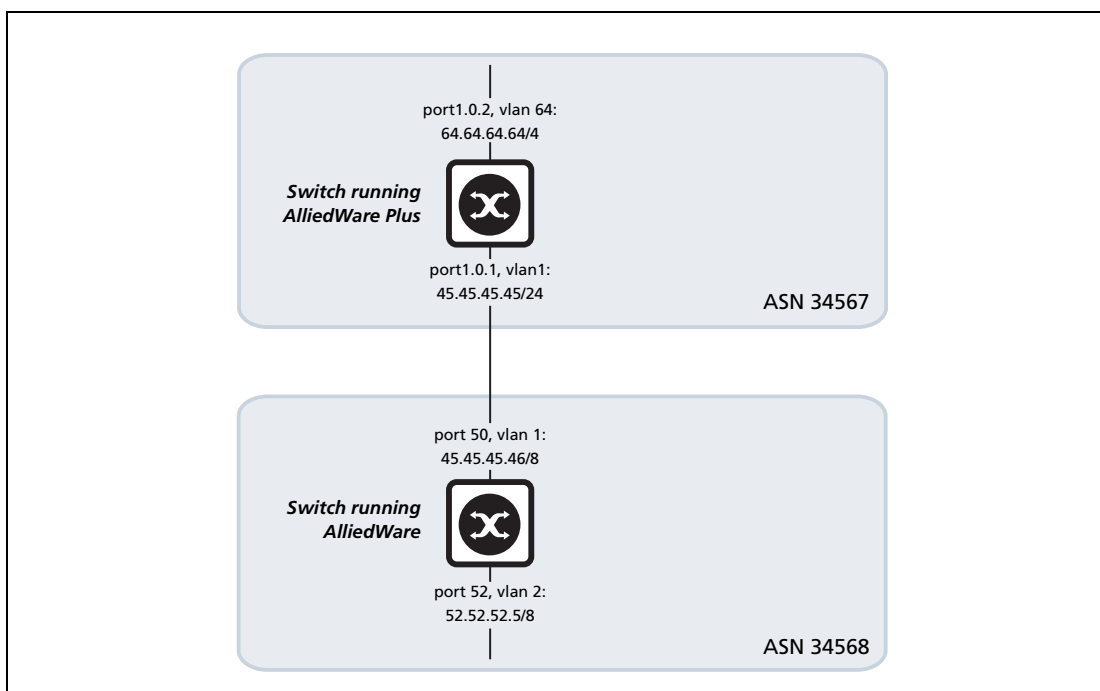
However, route maps are applied last, after the other types of filter. This is because route maps can modify updates, not just accept or discard them.

BGP: Example

This How To Note illustrates each type of filter with an example from a very simple BGP network. This section describes the basic network configuration. The following sections add filtering options to it:

- ["Example: Distribute filters" on page 11](#)
- ["Example: AS path filters" on page 14](#)
- ["Example: Prefix filters" on page 19](#)

The network consists of two BGP peers in different ASes:



Basic configuration

This configuration gets the neighbor relationship established and some routes exchanged.

AlliedWare Plus switch Create the second VLAN and associate port 1.0.2 with it; assign IP addresses; and configure BGP.

```
vlan database
vlan 64 name v64
interface port1.0.2
switchport access vlan 64
interface vlan1
ip address 45.45.45.45/24
interface vlan64
ip address 64.64.64.64/4
router bgp 34567
redistribute connected
neighbor 45.45.45.46 remote-as 34568
```

AlliedWare switch Create the second VLAN and associate port 52 with it; assign IP addresses; and configure BGP.

```
create vlan="v2" vid=2
add vlan="2" port=52
enable ip
set ip autonomous=34568
add ip int=vlan1 ip=45.45.45.46
add ip int=vlan2 ip=52.52.52.5
set bgp ro=45.45.45.46
add bgp pe=45.45.45.45 rem=34567
ena bgp pe=45.45.45.45
add bgp imp=interface
```

Note: the prefix lengths on the UIP addresses on VLAN1 of the two switches are different. The VLAN1 address on the AlliedWarePlus switch has prefix length 24, and that on the AlliedWare switch has prefix length 8. Although this is a non-standard, and not recommended, configuration, it has been done deliberately for the purposes of the illustrative examples in this How To Note. By having different prefix lengths on those addresses, it is clear in the route tables of the two switches which of the 45.x.x.x routes is a connected route, and which is a BGP-learned route.

Confirming the neighbor relationship

Check that each switch sees the interface route advertised from the other switch. On both the AlliedWare Plus and AlliedWare switches, use the command **show ip route**.

AlliedWare Plus switch

```
awplus#show ip route
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:18:10
C       45.45.45.0/24 is directly connected, vlan1
B       52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:18:10
C       64.0.0.0/4 is directly connected, vlan64
```

AlliedWare switch

```
Manager BGP peer> show ip route
```

```
IP Routes
-----
```

Destination	Mask Type	Policy	NextHop Protocol	Flags Tag	Interface Metrics	Age Pref
45.0.0.0	255.0.0.0		0.0.0.0	-----	vlan1	2128
	direct	0	interface	-	1	0
45.45.45.0	255.255.255.0		45.45.45.45	-----	vlan1	993
	remote	0	bgp	-	2	170
52.0.0.0	255.0.0.0		0.0.0.0	-----	vlan2	1642
	direct	0	interface	-	1	0
64.0.0.0	240.0.0.0		45.45.45.45	S----	vlan1	993
	remote	0	bgp	-	2	170

```
-----
```

BGP: Configuring distribute filters

Distribute filters use ACLs (Access Control Lists) to filter particular routes on the basis of their prefixes.

Distribute filters and prefix filters both filter individual routes out of BGP update packets. They are mutually exclusive.

About ACLs

From the point of view of route filtering, an ACL is one or more simple unnumbered filter entries, each with a prefix and an action of **deny** or **permit**.

You can use any of the following syntax options to create the ACL entries. The main difference is in how you label the ACL—whether you use a name or a number.

```
access-list standard <name> {deny|permit} <ipadd/prefixlength>
    exact-match
access-list <1-99> {deny|permit} <ipadd> <reverse-mask>
access-list <1300-1999> {deny|permit} <ipadd> <reverse-mask>
```

Entries are unnumbered, so each new entry gets added to the end of the ACL.

Named ACLs Using a standard named ACL lets you specify whether the prefix needs to be an exact match or not. If you specify **exact-match**, then routes only match the ACL if they have the specified prefix length. Otherwise, routes match the ACL if they have a prefix length equal to or longer than the specified prefix length. For example, if you specify 10.0.0.0/8, then:

- without **exact-match**, the ACL matches all of 10.0.0.0/8–10.0.0.0/32
- with **exact-match**, the ACL only matches 10.0.0.0/8

Numbered ACLs For numbered ACLs, the mask is a reverse (or wildcard) mask. This is the opposite of a standard mask in dotted decimal notation. However—in line with industry standards—the mask value has no effect. The ACL always applies to all prefix lengths.

Extended ACLs You can also use an extended ACL (number range 100-199, or 2000-2699, or by using the **extended <name>** parameter) but there is no advantage to doing so. Extended ACLs include two prefixes (source and destination), and using two prefixes is meaningless when filtering routes.

Using ACLs as filters

When you have created an ACL, you can use it to filter incoming or outgoing update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming updates (received from a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> distribute-list <acl-id> in
```

Filter outgoing updates (destined for a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> distribute-list <acl-id> out
```

The switch will then compare the prefixes in update packets with each entry in the ACL, looking for matches.

If a matching entry has the parameter **permit**, then there will be effectively no action. If a matching entry has the parameter **deny**, then the specified prefix will be removed from the update packet.

Once the update packet has been compared against every entry in the ACL, it will be sent to the neighbor (**out** filters) or processed (**in** filters), minus any prefixes that have been removed by the filter.

Example: Distribute filters

Filter out one particular route from a neighbor

This example expands on the basic configuration in "[BGP: Example](#)" on page 7.

It creates an ACL on the AlliedWare Plus switch that explicitly denies one of the routes that is advertised from the AW neighbour, and explicitly accepts all other routes.

1. Create a named ACL to deny the route 52.0.0.0/8 and accept all others. You need to include a **permit any** entry because ACLs end in an implicit **deny any** entry.

```
awplus(config)# access-list standard list1 deny 52.0.0.0/8 exact
awplus(config)# access-list standard list1 permit any
```

2. Set that ACL as the filter for the BGP neighbour 45.45.45.46:

```
awplus(config)# router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 distribute-list list1 in
```

3. Renew the route exchange by shutting down the neighbour, and then bringing it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

4. Check that the IP route table no longer includes 52.0.0.0/8:

```
awplus(config-router)# do show ip route

Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:14
C       45.45.45.0/24 is directly connected, vlan1
C       64.0.0.0/4 is directly connected, vlan64
```

Filter out a range of prefix lengths

This example demonstrates the effect of the **exact** parameter in the ACL by discarding all routes to 52.0.0.0 with prefix lengths of 4 or greater.

1. Remove the existing ACL:

```
awplus(config)# no access-list standard list1
```

2. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

3. Check that the IP route table now includes all the routes:

```
awplus(config-router)#do show ip route

Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:01:57
C       45.45.45.0/24 is directly connected, vlan1
B       52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:01:57
C       64.0.0.0/4 is directly connected, vlan64
```

4. Add the following ACL entries. When **exact** is not specified, the ACL entry matches all masks greater than or equal to the specified mask, so the first entry below includes 52.0.0.0/8.

```
awplus(config)#access-list standard list1 deny 52.0.0.0/4
awplus(config)#access-list standard list1 permit any
```

5. Shut down the neighbor, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

6. Check that the IP route table no longer includes 52.0.0.0/8:

```
awplus(config-router)#do show ip route

Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:08
C       45.45.45.0/24 is directly connected, vlan1
C       64.0.0.0/4 is directly connected, vlan64
```

Use a numbered ACL instead of a named ACL

This example demonstrates a numbered ACL by discarding all routes to 52.0.0.0.

1. Create a numbered ACL:

```
awplus(config)#access-list 1301 deny 52.0.0.0 0.0.0.255
awplus(config)#access-list 1301 permit any
```

In line with industry standards, the wildcard mask is required but its value has no effect. The ACL always applies to all prefix lengths.

2. Set that ACL as the filter for the BGP neighbour 45.45.45.46:

```
awplus(config)#router bgp 34567
awplus(config-router)#neighbor 45.45.45.46 distribute-list 1301 in
```

3. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)#neighbor 45.45.45.46 shutdown
awplus(config-router)#neighbor 45.45.45.46 no shutdown
```

4. Check that the IP route table no longer includes 52.0.0.0/8:

```
awplus(config-router)#do show ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:05:30
C       45.45.45.0/24 is directly connected, vlan1
C       64.0.0.0/4 is directly connected, vlan64
```

BGP: Configuring AS path filters

To configure path filters we need to first understand something about AS path lists and how to use them.

AS path lists

Path filters use a construct known as an **AS path list**. An AS path list has a name and consists of one or more (unnumbered) entries. Each entry specifies:

- which AS paths to consider
- whether the AS paths in question should be included or excluded from the list.

The set of paths to consider is specified using regular expressions. The AlliedWare Plus OS supports the full set of syntax elements for simple regular expressions:

Symbol	Character	Used to...
^	Caret	Match the beginning of the input string. When used at the beginning of a string of characters, it negates a pattern match.
\$	Dollar sign	Match the end of the input string.
.	Period	Match a single character (white spaces included).
*	Asterix	Match zero or more sequences of a pattern.
+	Plus sign	Match one or more sequences of a pattern.
?	Question mark	Match zero or one occurrences of a pattern.
_	Underscore	Match spaces, commas, braces, parentheses, or the beginning and end of an input string.
[]	Brackets	Specify a range of single-characters.
-	Hyphen	Separate the end points of a range.

For example, you can specify things like:

- any path that contains a certain set of AS numbers: e.g. 23334 45634 8988
- any path that starts with a particular AS number: e.g. ^23334
- any path that ends with a particular AS number: e.g. 8988\$
- a specific path: e.g. ^23334 45634 8988\$
- an empty path: ^\$

Syntax To create an entry in an AS path list, use the command:

```
awplus(config)# ip as-path access-list <list-name> {deny|permit}
<regular-expression>
```

There is an implicit “deny all” entry at the end of each path list. Any AS path that does not match any earlier entry is excluded from the list.

Using AS path lists as path filters

When an AS path list has been created, it can be applied to filter incoming or outgoing update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming updates (received from a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> filter-list <list-name> in
```

Filter outgoing updates (destined for a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> filter-list <list-name> out
```

The router will then compare the AS path attribute in BGP update packets with each entry in the AS path list until a match is found. If the AS path list entry that matches has the parameter **permit**, then the update packet will be allowed through by the filter. If the matching entry has the parameter **deny**, then the update packet will be blocked by the filter.

Note: All update packets whose AS paths do not explicitly match an entry in the AS path list will be dropped, because the list ends in an implicit “deny all” entry.

Example: AS path filters

Discard or allow routes from a neighbor

This example expands on the basic configuration in "[BGP: Example](#)" on page 7.

First, it creates an AS path filter on the AlliedWare Plus switch that explicitly includes only AS 23456. AS path filters end in an implicit deny clause, so this filter implicitly excludes

AS 34568. After demonstrating that AS 34568 is excluded, the example then adds an entry to the filter to explicitly allow AS 34568.

Implicitly exclude a neighbor

1. If you previously configured a distribute filter, as shown in ["Example: Distribute filters" on page 11](#), remove it from the neighbor definition:

```
awplus(config)#router bgp 34567
awplus(config-router)#no neighbor 45.45.45.46 distribute-list 1301 in
```

2. Create an AS path access list that includes AS 23456:

```
awplus(config)#ip as-path access-list list1 permit 23456
```

3. Set that access list as the in-filter for the BGP neighbour 45.45.45.46:

```
awplus(config)#router bgp 34567
awplus(config-router)#neighbor 45.45.45.46 filter-list list1 in
```

4. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)#neighbor 45.45.45.46 shutdown
awplus(config-router)#neighbor 45.45.45.46 no shutdown
```

5. Check that the IP route table does not have the BGP routes from the AlliedWare neighbour in AS 34568 any more:

```
awplus(config-router)#do show ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
* - candidate default
```

```
C      45.45.45.0/24 is directly connected, vlan1
C      64.0.0.0/4 is directly connected, vlan64
```

Explicitly include a neighbor

6. Add an entry to the an AS path access list that explicitly includes AS 34568:

```
awplus(config)#ip as-path access-list list1 permit 34568
```

7. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)#neighbor 45.45.45.46 shutdown
awplus(config-router)#neighbor 45.45.45.46 no shutdown
```

- Check that the IP route table has the BGP routes from the AlliedWare neighbour in AS 34568 again:

```
awplus(config-router)#do show ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B      45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:03
C      45.45.45.0/24 is directly connected, vlan1
B      52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:03
C      64.0.0.0/4 is directly connected, vlan64
```

- Check that the AS path list shows the two filter entries:

```
awplus(config-router)#do show ip as-path-access-list
```

```
AS path access list list1
  permit 23456
  permit 34568
```

An outgoing filter that uses an AS-path list

- Create an AS-PATH list that denies empty AS Paths, but allows AS Paths that contain the AS number 34567.

```
ip as-path access-list example deny ^$
ip as-path access-list example permit 34567
```

- Apply this as the out route map for neighbour 45.45.45.46

```
neighbor 45.45.45.46 filter-list example out
```

- Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

- There are no routes advertised to the other side, so that says that the AS Path in the update packets is not 34567 at the point where the updates are passed to the out-going filter.

- Now remove the deny entry for empty AS-paths, and add an allow entry for empty AS_paths:

```
(config)#no ip as-path access-list example deny ^$
(config)#ip as-path access-list example permit ^$
```

So, the as-path list becomes:

```
ip as-path access-list example permit 34567
ip as-path access-list example permit ^$
```

- Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

- Now the routes are received by the neighbour. So, that does demonstrate that the AS-paths in the updates are empty at the moment they are examined by the filter. But, in the

neighbour's BGP table, the AS Path for the learnt routes is shown as "SEQ 34567". So, this entry is inserted into the AS-Path of the updates after they have passed through the filter.

```

BGP route table
Flags: >=Best route for the given prefix, *=Unreachable next hop,
W=Withdrawn
      m=Community, a=Aggregate route, s=Aggregate Suppressed, D=Damped
      Learned from: L=Local, e=eBGP Peer, i=iBGP Peer, c=Confederate Peer
-----
Fl  Prefix                Next hop          Origin    MED      Local pref
   Path
   Originator            Cluster List
-----
> 45.0.0.0/8            0.0.0.0          IGP       -        100
   EMPTY
L -
> 45.45.45.0/24        45.45.45.45     INCOMPLETE -        100
   SEQ 34567;
e -
> 52.0.0.0/8            0.0.0.0          IGP       -        100
   EMPTY
L -
> 64.0.0.0/4            45.45.45.45     INCOMPLETE -        100
   SEQ 34567;
e -
-----

```

BGP: Configuring prefix filters

Prefix filters use prefix lists to filter particular routes on the basis of their prefixes.

Prefix filters and distribute filters both filter individual routes out of BGP update packets. They are mutually exclusive.

About prefix lists

A prefix list is a list of prefix entries. Each entry specifies a particular prefix, a mask length or range of mask lengths, and whether or not those prefixes are deemed to explicitly match or explicitly **not** match the prefix list.

A prefix list entry is created with the command:

```

awplus(config)# ip prefix-list <list-name> [seq <number>]
                    {deny|permit} {any|<ipadd>/<prefix-length>}
                    [ge <minimum-length>] [le <maximum-length>]

```

You can choose to give an entry a sequence number by using the optional **seq** parameter. If you do not, the switch assigns sequence numbers in steps of 5 (number 5, 10, 15 etc.) and puts the new entry at the end of the list of entries.

To see entries and their numbers, use the command:

```
awplus# show ip prefix-list
```

Mask length

You can specify a single prefix mask length, or you can use the **ge** and **le** parameters to specify a range of mask lengths for the entry to match.

If you set the mask length to:

- a **single** mask length (by specifying neither the **ge** nor the **le** parameter), then a route matches against this entry if its prefix mask length is exactly that length.
- a **range** of mask lengths, then a route matches against this entry if its prefix mask length is greater than or equal to **ge** and less than or equal to **le**.

For example, to deny the IP addresses between 10.0.0.0/14 (mask of 255.252.0.0) and 10.0.0.0/22 (mask of 255.255.252.0) within the 10.0.0.0/8 (mask of 255.0.0.0) addressing range, use the command:

```
awplus(config)# ip prefix-list mylist deny 10.0.0.0/8 le 22 ge 14
```

The mask length (8 in this example) must be less than the value specified for the **ge** parameter. However, in line with industry standards, the mask length has no other significance.

Using prefix lists as prefix filters

When you have created a prefix list, you can use it to filter incoming or outgoing update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming updates (received from a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> prefix-list <list-name> in
```

Filter outgoing updates (destined for a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> prefix-list <list-name> out
```

The router will then compare the prefixes in update packets with each entry in the prefix list, looking for matches.

If a matching entry has the parameter **permit**, then there will be effectively no action. If a matching entry has the parameter **deny**, then the specified prefix will be removed from the update packet.

Once the update packet has been compared against **every** entry in the prefix list, it will be sent to the neighbor (**out** filters) or processed (**in** filters), minus any prefixes that have been removed by the filter.

Example: Prefix filters

Filter out one particular route from a neighbor

This example expands on the basic configuration in "BGP: Example" on page 7.

It creates a prefix list on the AlliedWare Plus switch that explicitly permits one of the routes that is advertised from the AW neighbour, and therefore implicitly denies the other route.

1. If you previously configured an AS path filter, as shown in "Example: AS path filters" on page 14, remove it from the neighbor definition:

```
awplus(config)#router bgp 34567
awplus(config-router)#no neighbor 45.45.45.46 filter-list list1 in
```

2. Create an IP prefix list to include the route 45.0.0.0/8. Prefix lists end in an implicit exclude clause, so this will exclude the other route (52.0.0.0/8).

```
awplus(config)#ip prefix-list list1 permit 45.0.0.0/8
```

3. Set that prefix list as the prefix list filter for the BGP neighbour 45.45.45.46:

```
awplus(config)#router bgp 34567
awplus(config-router)#neighbor 45.45.45.46 prefix-list list1 in
```

4. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)#neighbor 45.45.45.46 shutdown
awplus(config-router)#neighbor 45.45.45.46 no shutdown
```

5. Check that the IP route table now contains one of the routes learnt from that neighbour, but not the other:

```
awplus(config-router)#do show ip route

Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:06
C       45.45.45.0/24 is directly connected, vlan1
C       64.0.0.0/4 is directly connected, vlan64
```

Filter out a range of different prefix lengths

This example filters out routes to 45.0.0.0 with prefix lengths in the range 5-8, accepts routes to 52.0.0.0/8, and implicitly filters out all other BGP routes.

1. Remove the existing prefix list entry:

```
awplus(config)#no ip prefix-list list1 permit 45.0.0.0/8
```

2. Add the following prefix list entries:

```
awplus(config)# ip prefix-list list1 deny 45.0.0.0/4 ge 5 le 8
awplus(config)# ip prefix-list list1 permit 52.0.0.0/8
```

3. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

4. Check that the IP route table now contains only the other route from the neighbour:

```
awplus(config-router)# do show ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default
```

```
C      45.45.45.0/24 is directly connected, vlan1
B      52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:26:45
C      64.0.0.0/4 is directly connected, vlan64
```

BGP: Configuring route maps

Route maps are very powerful and flexible entities. Therefore, the configuring of route maps must, by necessity, be relatively complex. The purpose of this section of the document is to understand route maps piece by piece and thereby build up a full understanding of how all the parts fit together.

Structure of a route map

There are various levels of structure within a route map:

- A route map is an entity with a name
- Each route map consists of multiple entries, identified by sequence numbers
- Each entry can consist of multiple clauses

In effect, an entry defines an individual filter. It can have a **match** clause that defines what it will match on, and it can have multiple **set** clauses that can specify actions to be taken.

An update packet is matched against each entry in turn. Once an entry is found that matches the packet, the action(s) associated with that entry is (are) performed, and no further entries are considered.

For example, if you create an entry that will permit an update packet, followed by an entry that would deny that packet, the packet is permitted. As another example, if you create two conflicting **set** clauses, the first change is applied, not the second.

There is an implicit “match all” filter at the end of the route map. The action on that implicit entry is **deny**. So, by default, any update packet that does not explicitly match any particular entry in the route map will be dropped.

You can change this by ending the route map with a “permit all” clause, such as the following:

```
awplus(config)# route-map <map-name> permit 65535
```

Clauses

There are two types of clauses that can be present in a route map entry:

- **match** clauses, which specify attributes or prefixes to match on
- **set** clauses, which specify the changes to be made to attribute values

A given route map entry can never have more than one match clause, but it can have multiple set clauses.

Configuring a match clause

When you configure a match clause, you can match on one of the attributes listed in the following sections.

An AS path list

For information about creating an AS path list, see ["AS path lists" on page 13](#).

Once you have made the path list, you can apply it to the match clause of a route map entry by using the commands:

```
awplus(config)# route-map <map-name> {deny|permit} <seq>
awplus(config-route-map)# match as-path <list-name>
```

This clause will cause the router to go through the entries in the AS path list. If one of the entries in the AS path list matches the AS path in the update packet, **and** the action on that AS path list entry is **permit**, then the packet is deemed to have matched this route map match clause.

The fact that there are two levels of matching going on, and effectively two different uses of the **permit/deny** parameters in the one clause, can be a bit confusing.

So, when working out the correct way to configure a clause that matches on AS path list, it is important to first think about how you are configuring the match criteria within the AS path list, and then separately think about the actions of the route map entry.

In particular, it is important to note that in this context, the parameters in the AS path list do **not** indicate whether the matching update packet (remember that the AS Path attribute is only set once in an update packet) is being allowed or dropped; "permit" and "deny" simply indicate whether the update packet is deemed to match or not match the AS path list. This is different to the case where the AS path list itself is being used as a filter.

Just as an example, consider these configurations:

Case 1:

```
awplus(config)# ip as-path access-list example deny ^$
awplus(config)# ip as-path access-list example permit 15557
awplus(config)# router bgp 100
awplus(config-router)# neighbor 192.168.200.201 filter-list example out
```

Case 2:

```
awplus(config)# ip as-path access-list example permit ^$
awplus(config)# ip as-path access-list example deny 15557
awplus(config)# route-map rmapexample deny 1
awplus(config-route-map)# match as-path example
awplus(config-route-map)# route-map rmapexample permit 65535
awplus(config-route-map)# router bgp 100
awplus(config-router)# neighbor 192.168.200.201 route-map rmapexample
out
```

Both of these configurations would cause outgoing update packets with empty AS paths to be dropped, and update packets with an AS path containing 15557 to be allowed.

But, to achieve that in the second case, the AS path list has to be configured to **permit** empty paths. This way, the empty path will match the AS path list, and be included into the route map's action of dropping packets that match the AS path list.

In the second case, the AS path list specifically excludes packets whose AS path contains 15557. Therefore, these packets are not dropped by the first route map entry, and are permitted by the last (permit all) route map entry.

A community list

Creating A community list is a set of entries that specify which community attribute values are included in or excluded from the list.

There are two types of community list: standard and expanded.

Standard lists are a just a list of one or more communities. They can be identified by a name, a number, or the word **standard**, and are created by using any of the following commands:

```
awplus(config)# ip community-list <1-99> {deny|permit} {aa:xx|internet|
local-as|no-advertise|no-export}
awplus(config)# ip community-list <list-name> {deny|permit} {aa:xx|
internet|local-as|no-advertise|no-export}
awplus(config)# ip community-list standard <list-name>
{deny|permit} {aa:xx|internet|local-as|
no-advertise|no-export}
```

If you have more than one community in an entry, separate them with spaces.

Expanded lists use regular expressions to specify the communities. They can be identified by a number, or by the word **expanded**, and are created by using any of the following commands:

```
awplus(config)# ip community-list <100-199> {deny|permit}
<reg-exp>
awplus(config)# ip community-list expanded <list-name>
{deny|permit} <reg-exp>
```

You can have multiple entries in a community list. Entries are unnumbered, so each new entry gets added at the end of the list.

There is an implicit “exclude all else” entry at the end of the community list.

Applying Once you have created a community list, use it in a route map entry by using the command:

```
awplus(config-route-map)# match community <list> [exact-match]
```

The optional **exact-match** parameter specifies that the communities contained in the attribute section of the update message must exactly match the specified community list. If

you do not specify **exact-match**, then the set of communities in the attribute list of the update message must include all the communities in the specified community list, but can also include other communities.

One or more prefixes, by using a prefix list

For information about creating a prefix list, see ["About prefix lists" on page 17](#).

Once you have made the prefix list, apply it to the match clause of a route map entry by using the command:

```
awplus(config-route-map)#match ip address prefix-list <list-name>
```

A prefix list can match a subset of prefixes in an update message. You can use this to change the attributes of some of the prefixes in an outgoing update, without having to change the attributes of all the prefixes. However, an update message contains just one set of attributes, which must apply to all the prefixes in the update.

So, if a route map entry matches on some of the prefixes in a particular outgoing update message, and has an associated set clause that specifies a change to the attributes for those prefixes, then the switch splits the update into two update packets:

- one that contains the original attribute values and the prefixes that were not included by the route map entry, and
- one that contains the new attribute values and the prefixes that were included by the route map entry.

One or more prefixes, by using an ACL

An ACL is an alternative to a prefix list for matching a prefix in an update message.

For information about creating an ACL, see ["About ACLs" on page 9](#).

Once you have made the ACL, apply it to the match clause of a route map entry by using the command:

```
awplus(config-route-map)#match ip address <acl-number-or-name>
```

A next hop address

You can use either a prefix list or an ACL to specify a next hop address. Once you have made the prefix list or ACL, apply it to the match clause of a route map entry by using one of the commands:

```
awplus(config-route-map)#match ip next-hop prefix-list <list-name>  
awplus(config-route-map)#match ip next-hop <acl-number-or-name>
```

An update message matches this route map entry if the next-hop attribute in the update equals an IP address permitted in the prefix list or ACL.

An origin

To match an origin, use the command:

```
awplus(config-route-map)#match origin {egp|igp|incomplete}
```

An update message matches this route map entry if the origin attribute in the update equals the origin specified in the entry.

A metric (the MED attribute)

To match the MED value, use the command:

```
awplus(config-route-map)#match metric <med-value>
```

An update message matches this route map entry if the MED attribute in the update equals the value specified in the entry.

Configuring a set clause

If a packet matches the **match** clause, then the action of the route map entry will be applied to that packet. The action might simply be to permit or deny the packet, or it might be to update its attributes by applying one or more **set** clauses.

Note: When configuring a set clause, make sure you are in route map mode for the same route map name sequence number as you used for the match clause. The prompt should look like:

```
awplus(config-route-map)#
```

A set clause can change any of the following attributes on an update message:

AS path Multiple autonomous system numbers (ASNs) can be added to the AS path attribute in the update packet. Use the command:

```
set as-path prepend <asn-list>
```

If adding multiple ASNs, separate them with spaces.

Community Community set clauses give you a lot of control over the community values in updates. You can:

- Replace the set of community values in an update, by using the command:

```
set community <community-values>
```
- Add more communities to the set of community values in an update, by using the command:

```
set community <community-values> additive
```
- Remove the community attribute from an update, by using the command:

```
set community none
```

- Remove communities from the set of community values in an update, by creating a community list (see "A community list" on page 23) and then applying it in a set clause. Use the command:

```
set comm-list <list-id>
```

Localpref This specifies a value to set as the Localpref attribute in the update message.

Use the command:

```
set local-preference <0-4294967295>
```

Metric (MED) This specifies a value to set as the MED attribute in the update message. You can:

- Set the MED in an update, by using the command:
- Increase or decrease the MED in an update by a specified amount, by using one of the commands:

```
set metric +<amount>
```

```
set metric -<amount>
```

For example, to increase the MED by 2, use the command:

```
set metric +2
```

If you want the switch to compare MED values in update messages from peers in different ASes, also enter the commands:

```
router bgp <asn>
```

```
bgp always-compare-med
```

The switch always compares MED values in update messages from peers in the same AS.

Origin This specifies a value to set as the Origin attribute in the update message.

Use the command:

```
set origin {igp|egp|incomplete}
```

Aggregator This specifies the update's aggregator attribute. The aggregator attribute specifies the AS and IP address of the device that performed the aggregation.

Use the command:

```
set aggregator <asn> <ipadd>
```

Atomic-aggregate This adds the atomic aggregate attribute to the update.

Use the command:

```
set atomic-aggregate
```

Extended community This specifies a value to set as the Extended Community attribute in the update message.

Use the command:

```
set extcommunity {rt|soo} <ext-comm-number>
```

The **rt** parameter configures a route target extended community. This consists of routers that will receive matching routes.

The **soo** parameter configures a site-of-origin extended community. This consists of routers that will inject matching routes into BGP.

Next hop This specifies the next hop for matching routes.

Use the command:

```
set ip next-hop <ipadd>
```

Originator ID This specifies the Originator ID attribute in the update message. The originator ID is the router ID of the IBGP peer that first learned this route, either via an EBGP peer or by some other means such as importing it.

Use the command:

```
set originator-id <ipadd>
```

Weight Specifies a weight value for matching routes. The weight value assists in best path selection of BGP routes. It is stored with the route in the BGP routing table, but is not advertised to peers. When there are multiple routes with a common destination, the device uses the route with the highest weight value.

Use the command:

```
set weight <0-4294967295>
```

Dampening This is used to turn out BGP dampening and optionally set dampening parameters for the routes that match the route map entry.

Use one of the commands:

```
set dampening
set dampening <reachtme>
set dampening <reachtme> <reuse> <suppress> <maxsuppress>
```

The effect of different combinations of clauses

A map entry could consist of:

- one match clause with an action, or
- no match clause and one or more set clauses, or
- one match clause and one or more set clauses

Let us consider each entry type in turn.

One match clause with an action

The effect of an entry that contains a match, but no sets, will be to apply the specified action to all update messages that match the entry. The action can be either permit or deny. So, this is simply a filter—it simply lets updates through or blocks them; it does not alter updates in any way.

If the match criterion in the match clause specifies AS path or Community and the action is deny, then whole update messages that match the criterion will be discarded.

If the match criterion in the match clause specifies a prefix list or ACL, then **only** the prefixes which are in the prefix list will be dropped. Those prefixes will be removed from the update message, but other prefixes in the update message will not be removed.

No match clause and one or more set clauses

If an entry has no match clause, then it is assumed to match **all** update packets.

It is possible to add one set clause for each type of attribute that can be set. Of course, it is not compulsory to add a clause for each of these attributes. If there is no set clause for an attribute, the switch will simply leave it unaltered in update packets.

Any attribute for which there is not a set clause added will be simply left unaltered in the update packets that are processed by the entry.

It is possible to specify a route map action of deny for a route map that includes a set clause, but then there is no point in having defined any attributes to be set, as the routes are going to be discarded anyway. However, this is a neat way of turning a clause on or off without destroying the configuration entirely.

Note: Once you have created an entry that has no match clause, there is no point in adding any more entries below that one, as this entry will match **all** packets, and the entries below it will never be considered.

A match clause and one or more set clauses

The most general case is for an entry to have one match clause (it can never have more than one match clause) and a number of set clauses.

Obviously enough, such an entry will apply the activities defined by the set clauses to only those update packets that are picked out by the match clause.

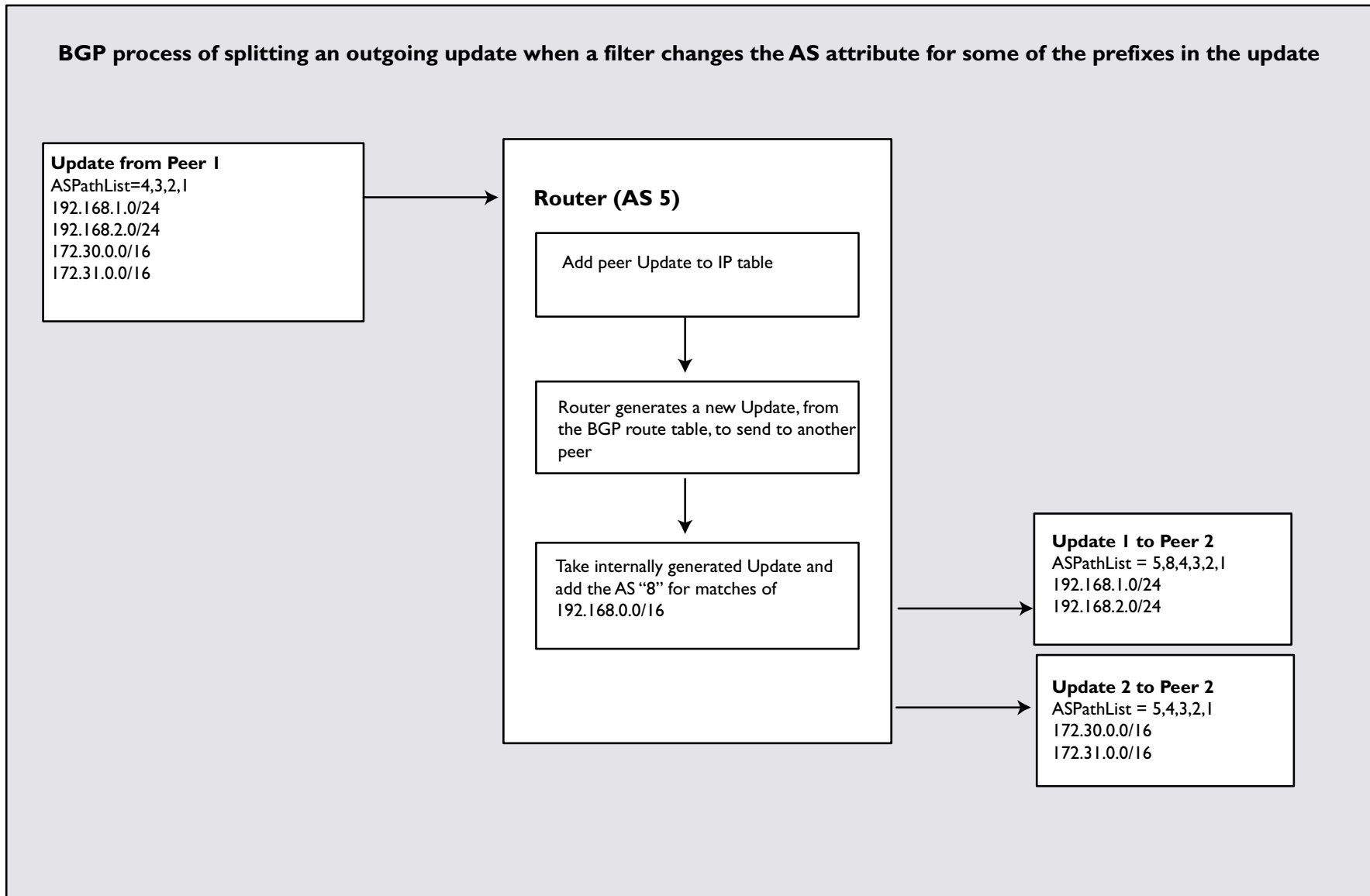
Particular mention, though, has to be made of the case where the match clause specifies prefix list or ACL as the match criterion, and the route map is being applied to outgoing route updates.

The intention of such an entry would be that the attribute values specified in the set clauses be applied to **only** those routes that are contained in the prefix list or ACL specified in the match clause.

So, what happens when there is an outgoing update message that contains several prefixes, some of which are in the match clause and some of which are not? Given that an update packet contains just one set of attributes, it is not possible to say “change some of the attributes in the packet, but make them apply to only some of the prefixes contained in the packet”. The attributes in an update packet must apply to **all** the prefixes in the packet.

The only solution is to break up the update packet. The switch creates one packet that contains the original attribute values and the prefixes that were not specified in the match clause of the route map entry. Then the switch creates another packet that contains those prefixes of the original packet which did match the prefix list or ACL in the match clause, but with modified attribute values.

The following diagram illustrates the BGP process of splitting an outgoing update:



BGP: Applying distribute, path, prefix, and route map filters to a peer

Distribute filters, path filters, prefix filters, and route maps can all be applied to a BGP peer configuration for both incoming and outgoing updates. However, you cannot combine distribute filters (ACLs) and prefix filters.

First, enter BGP router mode for the AS. The prompt should look like:

```
awplus(config-router)#
```

Then use the following commands.

To apply an **ACL** to filter incoming route updates:

```
neighbor <ipadd> distribute-list <acl> in
```

To apply a **path list** to filter incoming route updates:

```
neighbor <ipadd> filter-list <list-name> in
```

To apply a **prefix list** to filter incoming route updates:

```
neighbor <ipadd> prefix-list <list-name> in
```

To apply a **route map** to filter incoming route updates or alter their attributes:

```
neighbor <ipadd> route-map <map-name> in
```

To apply an **ACL** to filter outgoing route updates:

```
neighbor <ipadd> distribute-list <acl> out
```

To apply a **path list** to filter outgoing route updates:

```
neighbor <ipadd> filter-list <list-name> out
```

To apply a **prefix list** to filter outgoing route updates:

```
neighbor <ipadd> prefix-list <list-name> out
```

To apply a **route map** to filter outgoing route updates or alter their attributes:

```
neighbor <ipadd> route-map <map-name> out
```

You can also use an AS path list, prefix list or ACL in a route map, instead of applying it directly as a filter.

As mentioned in "[Hierarchy of the different filters](#)" on page 7, if you configure multiple types of filter, the switch applies the route maps last. This is true in both the incoming and outgoing directions.

Examples

Example A A route map that matches on a prefix-list and sets the route metrics

1. Create a prefix list that matches just 52.0.0.0/8

```
awplus(config)#ip prefix-list test1 permit 52.0.0.0/8
```

2. Then, create a route map to match on this prefix-list, and set the metric of matching routes to 665:

```
awplus(config)#route-map test1 permit 1
awplus(config-route-map)#match ip address prefix-list test1
awplus(config-route-map)#set metric 665
```

3. Remove any filters previously configured on neighbour 45.45.45.46, and configure the route map as an incoming filter:

```
awplus(config)#router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 route-map test1 in
```

4. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

5. Check the route table. You will see that the only route learnt from the peer is 52.0.0.0/8, and it has metric 665. The route 45.0.0.0/8 has been filtered out by the implicit deny-all entry at the end of the route map:

```
awplus#sh ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
O - OSPF, IA - OSPF inter area
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
E1 - OSPF external type 1, E2 - OSPF external type 2
* - candidate default
```

```
C    45.45.45.0/24 is directly connected, vlan1
B    52.0.0.0/8 [20/665] via 45.45.45.46, vlan1, 00:03:53
C    64.0.0.0/4 is directly connected, vlan64
```

Example B Match on a prefix-list that denies an entry

1. Create a prefix list that excludes 52.0.0.0/8, then includes all other routes:

```
awplus(config)#ip prefix-list test2 deny 52.0.0.0/8
awplus(config)#ip prefix-list test2 permit any
```

2. Then, set entry 1 of the route map test1 to match on this prefix-list:

```
awplus(config)#route-map test1 permit 1
awplus(config-route-map)#no match ip address prefix-list test1
awplus(config-route-map)#match ip address prefix-list test2
```

So, the route map is:

```
route-map test1, permit, sequence 1
Match clauses:
ip address prefix-list test2
```

```
Set clauses:
```

```
metric 665
```

3. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
```

```
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

4. Check the route table. This time, the only route learnt from the peer is 45.0.0.0/8, and it still has metric 665. The prefix-list has explicitly matched all routes except 52.0.0.0/8, so all routes except 52.0.0.0/8 have been permitted by the first entry of the route map. Then 52.0.0.0/8 has been denied by the implicit deny-all entry at the end of the route map.

```
awplus#sh ip route
```

```
Codes: C - connected, S - static, R - RIP, B - BGP
```

```
O - OSPF, IA - OSPF inter area
```

```
N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
```

```
E1 - OSPF external type 1, E2 - OSPF external type 2
```

```
* - candidate default
```

```
B    45.0.0.0/8 [20/665] via 45.45.45.46, vlan1, 00:00:02
```

```
C    45.45.45.0/24 is directly connected, vlan1
```

```
C    64.0.0.0/4 is directly connected, vlan64
```

Example C Match on a community list, and apply to a deny entry of a route map

1. Create a community list that has no community types specified:

```
ip community-list 1 permit
```

The updates from the BGP peer switch will match this community list, as they have no community attribute.

2. Create an IP access-list that matches ALL routes:

```
access-list 1 permit any
```

3. Create a route map that has two clauses: one which denies all updates matching the community list, and one which permits all else.

```
route-map com-deny-test deny 1
```

```
match community 1
```

```
route-map com-deny-test permit 2
```

```
match ip address 1
```

4. Apply this route map as the in route map on the neighbour 45.45.45.46

```
awplus(config-router)# neighbor 45.45.45.46 route-map com-deny-test in
```

5. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
```

```
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

6. The routes from the neighbour are now denied, as they match the community-list, and therefore the deny entry in the route map:

```
awplus#sh ip route
```

```

Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

C      45.45.45.0/24 is directly connected, vlan1
C      64.0.0.0/4 is directly connected, vlan64

```

Example D Matching on a next-hop prefix-list

1. Create a prefix-list that matches on the neighbour's IP address:

```
awplus(config)#ip prefix-list nh-test permit 45.45.45.46/32
```

2. Create a route map with a permit entry that matches on this prefix-list as a next-hop:

```
awplus(config)#route-map nh-test permit 1
awplus(config-route-map)#match ip next-hop prefix-list nh-test
```

3. Apply this route map as the in route map on the neighbour:

```
awplus(config)#router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 route-map nh-test in
```

4. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

5. All the routes are learnt OK from the neighbour:

```
awplus#sh ip route
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B      45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:03
C      45.45.45.0/24 is directly connected, vlan1
B      52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:00:03
C      64.0.0.0/4 is directly connected, vlan64

```

Example E Prepending AS numbers

1. Create a route map that prepends a list of AS numbers to the attribute field of outgoing updates:

```
awplus(config)#route-map as-list-test permit 1
awplus(config-route-map)#match ip address 1
awplus(config-route-map)#set as-path prepend 11 22 33 44 55 66 77 88 99
1010 1111 1212 1313 1414 1515 1616
```

2. Set this as the out route map for the neighbour 45.45.45.46

```
awplus(config-route-map)#router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 route-map as-list-test out
```

3. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

4. Now, check the routes learnt on the neighbour - note the long AS-path list:

```
Manager BGP peer> sh bgp route
```

```
BGP route table
Flags: >=Best route for the given prefix, *=Unreachable next hop,
W=Withdrawn
      m=Community, a=Aggregate route, s=Aggregate Suppressed, D=Damped
      Learned from: L=Local, e=eBGP Peer, i=iBGP Peer, c=Confederate Peer
-----
Fl Prefix                Next hop          Origin    MED      Local pref
Path
Originator              Cluster List
-----
<SNIP>

> 64.0.0.0/4            45.45.45.45      INCOMPLETE -          100
  SEQ 34567 11 22 33 44 55 66 77 88 99 1010 1111 1212 1313 1414 1515
  1616;
  e -                    -
-----
```

Example G A community-list based filter dropping incoming updates based on community number

1. Configure the AW peer to send out a community number 89:89

```
add ip routem=com entry=1 set commmun=89:89
set bgp peer=45.45.45.45 out routemap=com sendcommunity=yes
```

So, the routes coming from that peer has community 89:89

```
BGP#show ip bgp 52.0.0.0/8
BGP routing table entry for 52.0.0.0/8
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  34568
    45.45.45.46 from 45.45.45.46 (45.45.45.46)
      Origin IGP metric 0, localpref 100, valid, external, best
      Community: 89:89 -----
      Last update: Wed Oct 1 06:47:01 2008
```

2. Then, on the AW+ switch, create a community-list that matches on 89:89 (among others).

```
ip community-list 78 permit 55:66
ip community-list 78 permit 89:89
ip community-list 78 permit 9999:89
```

3. Create a prefix-list that matches the route 52.0.0.0/8

```
ip prefix-list test1 permit 52.0.0.0/8
```

4. Set the community-list to be the match criterion on a deny entry in a route map

```
route-map com deny 1
  match community 78
```

5. Set the prefix-list to be the match criterion on a permit entry in the route map:

```
route-map com permit 2
  match ip address prefix-list test1
```

So, if the route 52.0.0.0/8 is dropped by this route map, we can be sure that it was dropped by the first deny entry, and not by the implicit deny-all entry at the end of the route map.

6. Apply this route map as the incoming filter for the neighbour 45.45.45.46:

```
neighbor 45.45.45.46 route-map com in
```

7. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

If the deny clause in the routemap does its job, then the routes advertised from the peer will be dropped. And, indeed they are dropped.

```
BGP#sh ip route
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

C       45.45.45.0/24 is directly connected, vlan1
C       64.0.0.0/4 is directly connected, vlan64
```

Example H Combination of a distribute list and a route map and an AS-Path list filter

1. Add some new static routes to the AW peer, and configure the peer to redistribute these static routes into BGP:

```
add ip rou=156.23.4.0 mask=255.255.255.248 int=vlan2 next=52.34.5.4
add ip rou=156.23.4.8 mask=255.255.255.248 int=vlan2 next=52.45.45.45
add ip rou=156.23.4.32 mask=255.255.255.224 int=vlan2 next=52.45.45.45
add ip rou=156.23.4.144 mask=255.255.255.240 int=vlan2 next=52.45.45.45
add bgp import=static
```

2. Configure the peer to give Community 89:89 to the route 156.23.4.32/27 and append AS path 34599 to the route 156.34.4.144/28.

```
add ip prefixlist=com entry=1 action=match prefix=156.23.4.32
  masklength=27
add ip prefixlist=as entry=1 action=match prefix=156.23.4.144
  masklength=28

add ip routem=mixed entry=1 match prefixlist=com
add ip routem=mixed entry=1 set com=89:89
add ip routem=mixed entry=2 match prefixlist=as
add ip routem=mixed entry=2 set aspath=34599

set bgp peer=45.45.45.45 outroutemap=mixed sendcommunity=yes
```

When the route 156.23.4.144/28 is checked on the AW+ switch, its ASPath contains 34599.

```
BGP#sh ip bgp 156.23.4.144/28
BGP routing table entry for 156.23.4.144/28
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
    34568 34599
      45.45.45.46 from 45.45.45.46 (45.45.45.46)
        Origin incomplete metric 0, localpref 100, valid, external, best
        Last update: Wed Oct 1 08:54:10 2008
```

3. On the AW+ switch, create an ASPath-list that drops AS paths containing 34599, but allows all others.

```
ip as-path access-list list1 deny 34599
ip as-path access-list list1 permit .*
```

4. Create a community-list that includes the community 89:89, among others.

```
ip community-list 78 permit 55:66
ip community-list 78 permit 89:89
ip community-list 78 permit 9999:89
```

5. Create a route map that drops updates that match this community-list, but allows all other updates.

```
route-map com deny 1
  match community 78
route-map com permit 2
```

6. Apply both the aspath-list and the route map as in-filters on the neighbour:

```
router bgp 34567
  redistribute connected
  neighbor 45.45.45.46 remote-as 34568
  neighbor 45.45.45.46 route-map com in
  neighbor 45.45.45.46 filter-list list1 in
```

With this combination, neither 156.23.4.32/27 nor 156.34.4.144/28 appear in the IP route table. The route 156.23.4.32/27 is dropped by the route map filter, and the route 156.34.4.144/28 is dropped by the ASPath-list filter.

```
BGP#sh ip route
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:06:39
C       45.45.45.0/24 is directly connected, vlan1
B       52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:06:39
C       64.0.0.0/4 is directly connected, vlan64
B       156.23.4.0/29 [20/0] via 45.45.45.46, vlan1, 00:06:39
B       156.23.4.8/29 [20/0] via 45.45.45.46, vlan1, 00:06:39
```

7. Now create an ACL that drops the route 156.23.4.0/28, and allows all others.

```
access-list standard list3 deny 156.23.4.0/28
access-list standard list3 permit any
```

8. Add that ACL as a distribute-list in-filter on the neighbour:

```
router bgp 34567
 redistribute connected
 neighbor 45.45.45.46 remote-as 34568
 neighbor 45.45.45.46 distribute-list list3 in
 neighbor 45.45.45.46 route-map com in
 neighbor 45.45.45.46 filter-list list1 in
```

9. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

Then all the 156.23.4.x routes are filtered out. The distribute-list filter drops both the routes 156.23.4.0/29 and 156.23.4.8/29 because the effect of the distribute-list filter is to drop all routes within the address range covered by 156.23.4.0/28, that have a prefix-length of 28 or longer.

```
BGP#sh ip route
Codes: C - connected, S - static, R - RIP, B - BGP
       O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       * - candidate default

B       45.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:16:21
C       45.45.45.0/24 is directly connected, vlan1
B       52.0.0.0/8 [20/0] via 45.45.45.46, vlan1, 00:16:21
C       64.0.0.0/4 is directly connected, vlan64
```

Example I Using an ACL-match in a route map to update just a single route out of an update**1. Configure static routes on the AW+ switch.**

```
BGP(config)#ip route 192.34.23.0/26 64.93.23.1
BGP(config)#ip route 192.34.23.32/29 64.193.54.1
BGP(config)#ip route 192.34.23.192/27 64.12.89.1
```

2. Configure BGP to redistribute static routes.

```
router bgp 34567
 redistribute connected
 redistribute static
 neighbor 45.45.45.46 remote-as 34568
```

3. Create an ACL that matches just one of these routes.

```
access-list standard marker permit 192.34.23.32/29
```

4. Create a route map with one clause that matches on this ACL and prepends an AS-path (and then has another clause the permits everything else).

```
route-map marker permit 1
 match ip address marker
 set as-path prepend 7822

route-map marker permit 2
```

5. Apply this route map as the out route map for the peer.

```
router bgp 34567
 redistribute connected
 redistribute static
 neighbor 45.45.45.46 remote-as 34568
 neighbor 45.45.45.46 route-map marker out
```

6. Shut down the neighbour, and then bring it up again:

```
awplus(config-router)# neighbor 45.45.45.46 shutdown
awplus(config-router)# neighbor 45.45.45.46 no shutdown
```

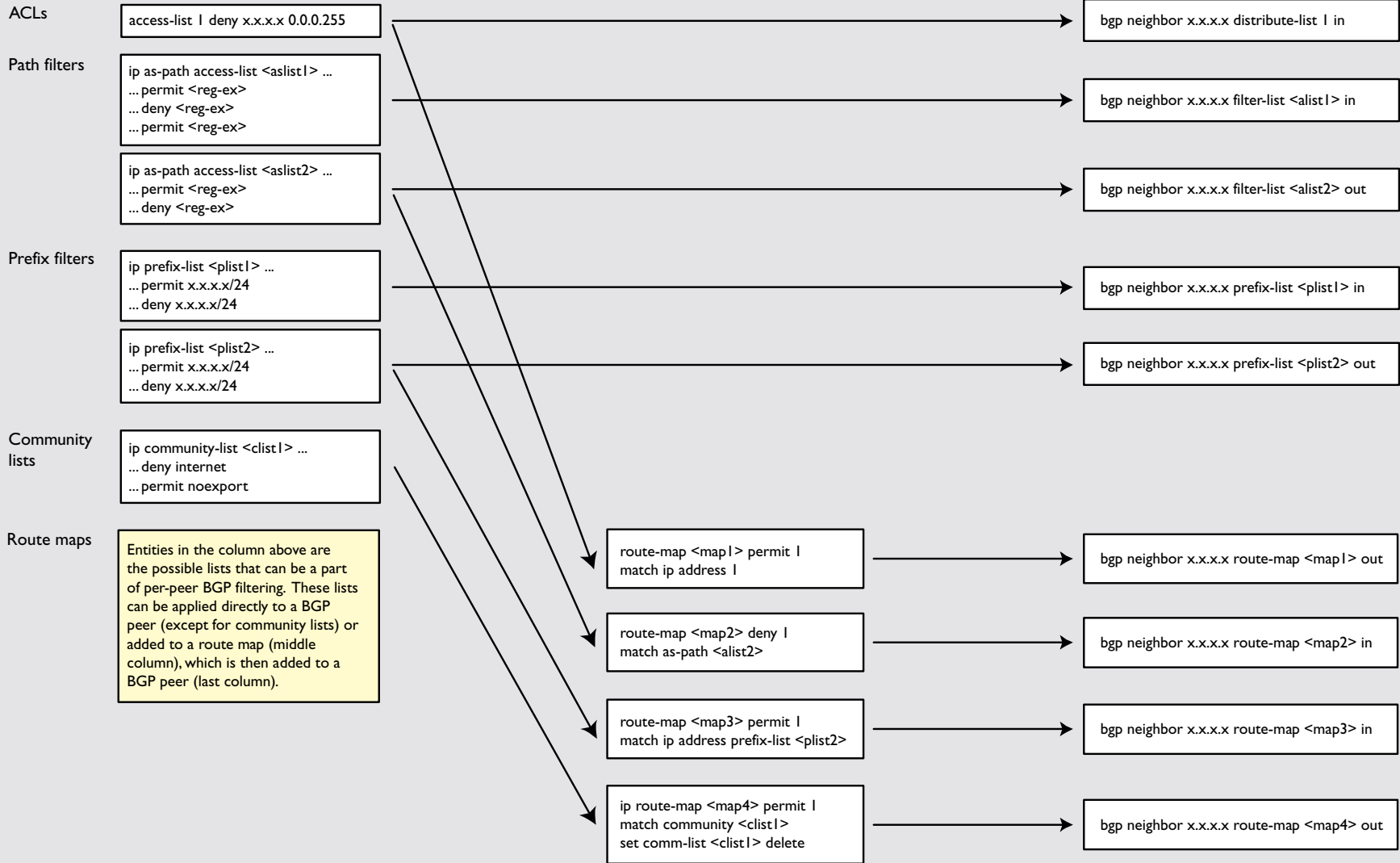
7. Check the BGP table on the peer. Only one route should arrive at the other end with that extra entry in the as-path.

```
BGP route table
Flags: >=Best route for the given prefix, *=Unreachable next hop,
W=Withdrawn
      m=Community, a=Aggregate route, s=Aggregate Suppressed, D=Damped
      Learned from: L=Local, e=eBGP Peer, i=iBGP Peer, c=Confederate Peer
-----
```

Fl	Prefix Path Originator	Next hop Cluster List	Origin	MED	Local pref
>	45.0.0.0/8 EMPTY	0.0.0.0	IGP	-	100
L	-	-			
>	45.45.45.0/24 SEQ 34567;	45.45.45.45	INCOMPLETE	-	100
e	-	-			
>	52.0.0.0/8 EMPTY	0.0.0.0	IGP	-	100
L	-	-			
>	64.0.0.0/4 SEQ 34567;	45.45.45.45	INCOMPLETE	-	100
e	-	-			
>	156.23.4.0/29 EMPTY	52.34.5.4	INCOMPLETE	-	100
L	-	-			
>	156.23.4.8/29 EMPTY	52.45.45.45	INCOMPLETE	-	100
L	-	-			
>	156.23.4.32/27 EMPTY	52.45.45.45	INCOMPLETE	-	100
L	-	-			
>	156.23.4.144/28 EMPTY	52.45.45.45	INCOMPLETE	-	100
L	-	-			
>	192.34.23.0/26 SEQ 34567;	45.45.45.45	INCOMPLETE	-	100
e	-	-			
>	192.34.23.32/29 SEQ 34567 7822;	45.45.45.45	INCOMPLETE	-	100
e	-	-			
>	192.34.23.192/27 SEQ 34567;	45.45.45.45	INCOMPLETE	-	100
e	-	-			

```
-----
```

Ways to use lists in IP route filtering for BGP, with generic command examples



Entities in the column above are the possible lists that can be a part of per-peer BGP filtering. These lists can be applied directly to a BGP peer (except for community lists) or added to a route map (middle column), which is then added to a BGP peer (last column).

BGP: Applying route maps to imported routes

The switch is able to import routes into BGP that it learnt by non-BGP means. In other words, the routes would be static routes, or routes learnt by OSPF or RIP.

You can apply a route map to this importation process so that the imported routes are given certain attributes, or so that certain routes are blocked from being imported.

The switch uses the route map to:

- filter routes or set attributes when it imports the routes into BGP
- set attributes on any update message in which it advertises the routes

Note that the entries in route maps that are applied to BGP importing cannot have match clauses that match on BGP attributes such as AS path or community. This is because BGP attributes are not relevant to non-BGP routes. The route map entries can match on prefix list, origin, or next hop, or the route type if importing OSPF routes.

Syntax

You can apply the route map by using the following commands in BGP router mode for the AS.

```
redistribute {connected|ospf|rip|static} route-map <map-name>
```

Other uses of route maps

Route maps are used in some contexts other than filtering routes. Let us look briefly at some of the other contexts in which they are used.

neighbor default-originate

The command **neighbor default-originate** instructs BGP to send a default route to a neighbour. This command includes a parameter for specifying a route map.

The **route map** parameter specifies criteria that must be fulfilled before the switch will advertise the default route to the neighbour. This lets you configure the switch so that it only acts as the default gateway for a neighbour if it has learned a route that makes it a suitable default gateway for that neighbour.

Specifically, the switch must have learnt a route that matches the permit criteria in the route map. If the switch does not know of any routes that match the permit criteria in the route map, then it will not advertise the default route.

neighbour unsuppress-map

When the **aggregate-address** command is used with the **summary-only** option, the more specific routes of the aggregate are suppressed to all neighbors. If you want to selectively leak some more-specific routes to a particular neighbour, then this is achieved by using the command **neighbor unsuppress-map**.

A required parameter on this command is the **route map** that specifies the set of more-specific routes to leak. Any route that matches permit criteria in the route map will be leaked.

network

The **network** command is used in BGP to selectively redistribute particular routes. Instead of saying “redistribute all static routes” or “redistribute all connected routes”, the **network** command says “redistribute this particular route”. Of course, the route has to be in the routing table before it can be redistributed, so the **network** command does not say “always import this route in BGP”, it says “*if this route is in the routing table, then import it into BGP*”.

As with other acts of redistribution, the **network** command has an optional **route map** parameter. This lets you set attributes on the route advertised by the **network** command.

You could use the route map for filtering the network, but it would be a little pointless to even configure the redistribution of this network if the associated route map simply dropped the route.

show ip bgp

The **show ip bgp** command takes an optional **route map** parameter. The effect of specifying the route map on the command is that the output shows information only on routes that match permit criteria on the route map.

BGP: Route map filtering example

Here is an example of a set of route maps.

BGP configuration

First, we need to set the router's ASN and the ASN of the peer.

```
router bgp 3816
neighbor 172.26.1.1 remote-as 15557
```

Neighbours are enabled by default.

Route map configuration

Next, we want to limit the routes that we accept from this peer. We will accept the default route from any community, but will only accept any other updates from community 15557.

```
ip prefix-list plist1 seq 1 permit 0.0.0.0/0
ip community-list 1 permit 15557
```

```
route-map accDef15557 permit 1
match ip address prefix-list plist1
route-map accDef15557 permit 2
match community 1 exact-match
```

```
router bgp 3816
neighbor 172.26.1.1 route-map accDef15557 in
```

Next, we want to change the attributes of certain routes before advertising them to the neighbor 172.26.1.1. For 192.168.2.0/24, we will add to the AS path. For 192.168.3.0/24 and 192.168.4.0/24, we will add a different set of entries to the AS path, and set the community and the local preference. All other routes will be advertised unchanged this neighbor.

```
ip prefix-list plist2 seq 1 permit 192.168.2.0/24
ip prefix-list plist3 seq 1 permit 192.168.3.0/24
ip prefix-list plist4 seq 1 permit 192.168.4.0/24
```

```
route-map outdef permit 1
match ip address prefix-list plist2
set as-path prepend 1645
```

```
route-map outdef permit 2
match ip address prefix-list plist3
set as-path prepend 65501 63901 43312
set community 0:23445 0:33445 no-advertise
set local-preference 9000
```

```

route-map outdef permit 3
match ip address prefix-list plist4
set as-path prepend 64751 1827
set community no-export additive
set local-preference 9000

route-map outdef permit 4

router bgp 3816
neighbor 172.26.1.1 route-map outdef out

```

Finally, we want to import the host-specific prefix 192.168.200.200/32 into BGP, and set a community value on that entry.

```

route-map netspec permit 1
set community 8888:8888 9999:9999
router bgp 3816
network 192.168.200.200/32 route-map netspec

```

OSPF: Configuring route maps for filtering and modifying OSPF routes

For information about route maps and their structure, see ["Structure of a route map" on page 21](#).

Route maps can be applied to OSPF routes as well as BGP routes. Of course, the route maps that can be used with OSPF are rather limited in comparison with those that are used with BGP, as OSPF route updates do not carry attributes in the way that BGP route updates do.

Also, OSPF route maps can **only** be applied to importing:

- OSPF-learned routes into the main IP route table, or
- static, BGP or RIP routes into OSPF

Filtering **cannot**:

- remove an entry from the LSA database once the entry has been added
- prevent the router from advertising an entry to interfaces in the same area that the entry is relevant to
- prevent updates that OSPF learns from being put into the LSA database
- change the properties of an entry in the LSA database

This is because OSPF shares LSAs between all the routers in an area. The protocol assumes that all the routers in the area have shared all the advertisements among each other, and that all agree on the state of the complete link state database for the area. If some routers in the area are learning, but not advertising, that breaks the OSPF model.

Configuring a match clause

A match clause can match on the following clauses.

Note: When configuring a match clause, make sure you are in route map mode.

The prompt should look like:

```
awplus(config-route-map)#
```

Metric

The entry will match all routes whose metric is equal to that specified in the clause.

To match a metric value, use the command:

```
match metric <value>
```

Interface

The entry will match all routes learnt via the specified VLAN.

To match a VLAN, use the command:

```
match interface <vlan>
```

External route type

The entry will match all routes of either Type 1 External or Type 2 External.

To match a route type, use the command:

```
match external {type-1|type-2}
```

A prefix, by using a prefix list

The entry will match one or more route prefixes.

For information about creating a prefix list, see ["About prefix lists" on page 17](#).

Once you have made the prefix list, apply it to the match clause of a route map entry by using the command:

```
match ip address prefix-list <list-name>
```

A prefix, by using an ACL

An ACL is an alternative to a prefix list for matching route prefixes.

For information about creating an ACL, see ["About ACLs" on page 9](#).

Once you have made the ACL, apply it to the match clause of a route map entry by using the command:

```
match ip address <acl-number-or-name>
```

A next hop address

The entry will match the route's next hop.

You can use either a prefix list or an ACL to specify a next hop address. Once you have made the prefix list or ACL, apply it to the match clause of a route map entry by using one of the commands:

```
match ip next-hop prefix-list <list-name>
match ip next-hop <acl-number-or-name>
```

Configuring a set clause

If a route matches the **match** clause, then the action of the route map entry will be applied to that route. The action might simply be to permit or deny the route, or it might be to update its parameters by applying one or more **set** clauses.

Note: When configuring a set clause, make sure you are in route map mode for the same route map name sequence number as you used for the match clause.

The prompt should look like:

```
awplus(config-route-map)#
```

A set clause can alter the following parameters on a route.

Metric This changes the route metric. You can:

- Set the metric, by using the command:

```
set metric <0-4294967295>
```

- Increase or decrease the metric by a specified amount, by using one of the commands:

```
set metric +<amount>
set metric -<amount>
```

For example, to increase the metric by 2, use the command:

```
set metric +2
```

Next hop This specifies the next hop for matching routes.

Use the command:

```
set ip next-hop <ipadd>
```

Type This sets the route type to either Type 1 External or Type 2 External.

Use the command:

```
set metric-type {type-1|type-2}
```

OSPF: Applying route maps

To specify a route map to be applied to static, BGP, RIP or connected routes as they are imported to OSPF, use the commands:

```
router ospf
 redistribute {bgp|rip|connected|static} route-map <map-name>
```

Note that if you want to filter OSPF routes as they are imported into the main IP route table, you need to use a distribute filter instead of a route map.

Use commands like the following:

```
router ospf 88
 distribute-list list1 in
```