

How do prefix-lists work?

By **Brian McGahan, CCIE #8593**

Prefix-lists are used to match on prefix and prefix-length pairs. Normal prefix-list syntax is as follows:

```
ip prefix-list LIST permit w.x.y.z/len
```

Where w.x.y.z is your exact prefix
And where len is your exact prefix-length

“ip prefix-list LIST permit 1.2.3.0/24” would be an exact match for the prefix 1.2.3.0 with a subnet mask of 255.255.255.0. This does not match 1.2.0.0/24, nor does it match 1.2.3.4/32, nor anything in between.

When you add the keywords “GE” and “LE” to the prefix-list, the “len” value changes its meaning. When using GE and LE, the len value specifies how many bits of the prefix you are checking, starting with the most significant bit.

```
ip prefix-list LIST permit 1.2.3.0/24 le 32
```

This means:
Check the first 24 bits of the prefix 1.2.3.0
The subnet mask must be less than or equal to 32

This equates to the access-list syntax:

```
access-list 1 permit 1.2.3.0 0.0.0.255  
ip prefix-list LIST permit 0.0.0.0/0 le 32
```

This means:
Check the first 0 bits of the prefix 0.0.0.0
The subnet mask must be less than or equal to 32
This equates to anything

```
ip prefix-list LIST permit 0.0.0.0/0
```

This means:
The exact prefix 0.0.0.0, with the exact prefix-length 0.
This is matching a default route.

```
ip prefix-list LIST permit 10.0.0.0/8 ge 21 le 29
```

This means:
Check the first 8 bits of the prefix 10.0.0.0

The subnet mask must be greater than or equal to 21, and less than or equal to 29.

```
ip prefix-list CLASS_A permit 0.0.0.0/1 ge 8 le 8
```

This matches all class A addresses with classful masks. It means:

Check the first bit of the prefix, it must be a 0.

The subnet mask must be greater than or equal to 8, and less than or equal to 8. (It is exactly 8)

When using the GE and LE values, you must satisfy the condition:

$$\text{Len} < \text{GE} \leq \text{LE}$$

Therefore “ip prefix-list LIST permit 1.2.3.0/24 ge 8” is not a valid list.

What you can not do with the prefix-list is match on arbitrary bits like you can in an access-list. Prefix-lists cannot be used to check if a number is even or odd, nor check if a number is divisible by 15, etc... Bit checking in a prefix-list is sequential, starting with the most significant (leftmost) bit.

The way prefix lists work are you can specify a network and mask or a network and a range of masks. Specifying a network and mask is fairly simple:

```
ip prefix-list mylist seq 10 permit 172.16.25.0/24
```

This will allow (match) the exact network 172.16.25.0/24 to pass the list. However prefix lists can also specify a network with a range of masks. For example:

```
ip prefix-list mylist seq 10 permit 172.16.0.0/16 ge 24 le 26
```

This will take the entire class B network 172.16.0.0 (172.16.0.0/16) and pass only subnets with a /24, /25 or /26 mask (ge 24 le 26). So the exact network 172.16.0.0/16 would actually fail the list because it does not have a mask of /24, /25 or /26.

By default if you only specify "ge" then any subnet with a mask greater than or equal to the ge value will pass. That is, ge all the way up to /32. For example:

```
ip prefix-list mylist seq 10 permit 10.10.10.0/24 ge 28
```

This list specifies any subnet within the 10.10.10.0/24 range that has a mask of /28 or greater (255.255.255.240 to 255.255.255.255). Again, the exact subnet 10.10.10.0/24 would fail because it does not have a mask of /28 or greater.

By default if you only specify "le" then any subnet with a mask less than or equal to the le value but greater than or equal to the mask specified will pass. That is, le all the way down to the mask listed. For example:

```
ip prefix-list mylist seq 10 permit 10.64.0.0/16 le 23
```

This list specifies any subnet within the 10.64.0.0/16 range that has a mask between /16 and /23, inclusive (255.255.0.0 to 255.255.254.0). In this case the exact subnet 10.64.0.0/16 would pass because it has a mask in the range /16 to /23.

The "permit any any" in a prefix list is:

```
ip prefix-list mylist seq 200 permit 0.0.0.0/0 le 32
```

I recommend getting quite familiar with them because they are very powerful and actually not too bad to use once you get used to them!
Hope this helps - Rob.

[IP prefix-list](#)

ip prefix-list provides the most powerful prefix based filtering mechanism

Here is a quick little tutorial on Prefix-lists for you.

A normal access-list CANNOT check the subnet mask of a network. It can only check bits to make sure they match, nothing more. A prefix-list has an advantage over an access-list in that it CAN check BOTH bits and subnet mask - both would have to match for the network to be either permitted or denied.

For checking bits a prefix list ALWAYS goes from left to right and CANNOT skip any bits. A basic example would be this:

```
172.16.8.0/24
```

If there is only a / after the network (no le or ge) then the number after the / is BOTH bits checked and subnet mask. So in this case it will check the 24 bits from left to right (won't care about the last 8 bits) AND it will make sure that it has a 24 bit mask. BOTH the 24 bits checked and the 24 bit subnet mask must match for the network to be permitted or denied.

No we can do a range of subnet masks also that could be permitted or denied:

172.16.8.0/24 ge 25

If we use either the le or ge (or both le and ge) after the /, then the number directly after the / becomes ONLY bits checked and the number after the ge or le (or both) is the subnet mask. So in this case we are still going to check the first 24 bits of the network from left to right. If those match we are then going to check the subnet mask, which in this case can be GREATER THAN OR EQUAL TO 25 bits - meaning that as long as the first 24 bits of the network match the subnet mask could be 25,26,27,28,29,30,31,or 32 bits. They would all match.

We can also do:

172.16.8.0/24 le 28

Again this will check the first 24 bits of the network to make sure that they match. Then it will check to make sure that the subnet mask is LESS THAN OR EQUAL TO 28 bits. Now this isn't going to be 28 bits down to 0 bits, the subnet mask can't be any lower than the bits we are checking. So the valid range of subnet masks for this one would be 28 bits down to 24 bits (24,25,26,27,and 28). All of those would match.

We can also do both ge and le:

172.16.8.0/24 ge 25 le 27

Here again we are checking the first 24 bits to make sure they match. Then our subnet mask must be GREATER THAN OR EQUAL TO 25 bits LESS THAN OR EQUAL TO 27 bits. Meaning that 25,26,and 27 bit subnet masks would match.

Now for a couple of examples:

If we have the following networks:

172.16.8.0/28
172.16.8.16/28
172.16.8.32/28
172.16.8.48/28
172.16.8.64/28

We could permit all of these networks with on prefix-list statement:

172.16.8.0/24 ge 28 le 28

This will check the first 24 bits to make sure they match. All of these networks have 172.16.8 as the first 24 bits, and it won't care what is in the last 8 bits. Then it will check to make sure that the subnet mask is GREATER THAN OR EQUAL TO 28 bits LESS THAN OR EQUAL TO 28 bits - the only number that works for this is 28 bits. So the first 24 bits in the network must match and it has to have a 28 bit subnet mask. All 5 of our networks would match for this.

We could be even more precise with this and use:

172.16.8.0/25 ge 28 le 28

If we take a look at our 4th octets we will see that for all of them the 128 bit is off so we can check that bit also (25 bits total we are checking).

```
0 -- 0 0 0 0 0 0 0 0
16 - 0 0 0 1 0 0 0 0
32 - 0 0 1 0 0 0 0 0
48 - 0 0 1 1 0 0 0 0
64 - 0 1 0 0 0 0 0 0
```

This would be closer to permitting the 5 networks that we have.

We could also permit only the classful networks. The first thing that we need to do is figure out exactly what a classful network is.

For a class A network we know that it has to have an 8 bit mask and must be between 0 and 127 in the first octet. If we break down 0 and 127 we get:

```
0 --- 0 0 0 0 0 0 0 0
127 - 0 1 1 1 1 1 1 1
```

For the first octet of a class A network the first bit has to be a 0, it must be off. So we can do a prefix-list like this:

0.0.0.0/1 ge 8 le 8

In our first octet the first bit is a 0 (which is what it would need to be to be class A), with the /1 we have we are ONLY checking the first bit to make sure it's a 0 (meaning it would be a class A network 0 - 127). We are then making sure that this class A network actually has a class A subnet mask of 8 bits, and only 8 bits would match.

For the class B's we need to make sure that they have a 16 bit subnet mask and that they are in the range of 128 - 191 in the first octet. If we break down 128 and 191 we get:

```
128 - 1 0 0 0 0 0 0 0
191 - 1 0 1 1 1 1 1 1
```

The first two bits are what we are going to care about. We need to make sure that the first two bits in the first octet are 1 0 . The first number that we can use as our standard we are checking against is 128 - 128 has a 1 0 as the first two bits in its first octet.

128.0.0.0/2 ge 16 le 16

So we are checking the first two bits to make sure the network has a 1 0, meaning that it must be in the range of 128 - 191. We are then going to check to make sure that it has the classful 16 bit mask, and ONLY a 16 bit mask.

Finally we have the class C networks. Class C networks are in the range of 192 - 223 and they must have a 24 bit mask. If we break down 192 and 223 we get:

```
192 - 1 1 0 0 0 0 0 0
223 - 1 1 0 1 1 1 1 1
```

The first 3 bits in the first octet are what we care about. 192 would be the first number we can put in that first octet that will have 1 1 0 as its first 3 bits.

192.0.0.0/3 ge 24 le 24

We are going to check the first 3 bits of the octet and make sure that its 1 1 0 meaning that it has to be in the range of 192 - 223 being class C, then we are going to check to make sure it has a class C classful subnet of 24 bits.

Finally how to permit or deny any could be very helpful since a Prefix-list just like an Access-list has an implicit deny at the end:

0.0.0.0/0 le 32

This is 'any' for a prefix-list. It says check 0 bits; I don't care what any of the bits are. It also says that the subnet mask can be 32 bits or less (down to the number of bits we are checking) down to 0. So we aren't going to check any bits and the network can have a subnet mask of anything between 0 and 32 bits. This would be 'any'.

Now for your Prefix-list:

In the 3rd Octet we have 1, 4, and 5. We'll break these down to binary to see if we can summarize these into one line:

```
1 - 0 0 0 0 0 0 0 1
4 - 0 0 0 0 0 1 0 0
5 - 0 0 0 0 0 1 0 1
```

For a Prefix-list we need to go from the left to the right and we can't skip bits. So for these three networks we would need to stop at the 8 bit since it is the last bit from left to right that is the same. This would give us 3 bits that are different, or 8 possible networks. We only have 3 of the 8 possible networks and we should not permit or deny more than we actually have. We should be as specific as possible.

If we leave the 91.86.1.0/24 alone by itself it will give us a Prefix-list of:

91.86.1.0/24

This will check the first 24 bits from left to right to make sure that they match, and it will also check to make sure that it has a 24-bit subnet mask.

For the 4 and 5 networks we can permit or deny both of those with one line. If we take a look at 4 and 5 again we can see that all of the bit's match down to the 2 bit. This would leave 1 bit that doesn't match, which would give us 2 possible networks,

both of which we have.

The Prefix-list to permit or deny both 4 and 5 would be:

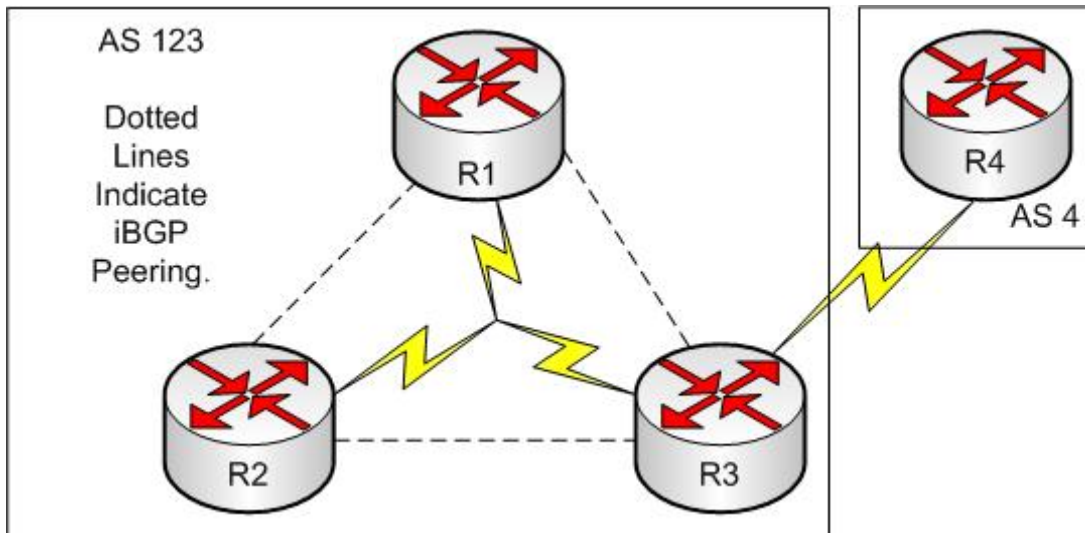
```
91.86.4.0/23 ge 24 le 24
```

This will check the first 23 bits from left to right. The 24th bit could either be off, which would give us 4, or it could be on which would give us 5. Since we have the ge and le involved the /23 is only bits checked. The ge and le specify that our subnet mask must be greater than or equal to 24-bits and less than or equal to 24-bits which means that the subnet mask must be 24-bits for both possible networks.

CCNP Certification BSCI Exam Tutorial: Using Prefix Lists To Filter BGP

By Chris Bryant, CCIE #12933

Once you have the fundamentals of BGP down, it's important to learn how to filter BGP routing updates. There are several methods of doing so, but the one Cisco recommends (and one you're sure to see plenty of on Cisco certification exams) is the use of prefix lists. The following network will be used in this tutorial to show the configuration and effect of prefix lists.



R4 is advertising three networks via BGP.

```
R4(config)#router bgp 4
R4(config-router)#network 21.0.0.0 mask 255.0.0.0
R4(config-router)#network 22.0.0.0 mask 255.0.0.0
R4(config-router)#network 23.0.0.0 mask 255.0.0.0
```

R4's eBGP neighbor R3 sees these routes and places them into its BGP table as shown below. R3 has two iBGP peers, R1 and R2, and is advertising itself as the next-hop IP address for all BGP routes sent to those two routers.

```
R3#show ip bgp
BGP table version is 4, local router ID is 3.3.3.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*> 21.0.0.0	10.2.2.4	0		0	4 i
*> 22.0.0.0	10.2.2.4	0		0	4 i
*> 23.0.0.0	10.2.2.4	0		0	4 i

```
R3(config)#router bgp 123
R3(config-router)#neighbor 172.12.123.1 next-hop-self
R3(config-router)#neighbor 172.12.123.2 next-hop-self
```

Both R2 and R1 see the three routes.

```
R2#show ip bgp
BGP table version is 4, local router ID is 2.2.2.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i21.0.0.0	172.12.123.3	0	100	0	4 i
*>i22.0.0.0	172.12.123.3	0	100	0	4 i
*>i23.0.0.0	172.12.123.3	0	100	0	4 i

```
R1#show ip bgp
BGP table version is 4, local router ID is 19.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i21.0.0.0	172.12.123.3	0	100	0	4 i
*>i22.0.0.0	172.12.123.3	0	100	0	4 i
*>i23.0.0.0	172.12.123.3	0	100	0	4 i

If we wanted R3 to receive all three of these routes from R4 but not advertise them to R2 and R1, we've got a couple of options on how to block these routes. Cisco's

recommendation is the use of prefix lists, and once you get used to the syntax (which you should do before taking and passing the BSCI), you'll see they are actually easier to use than access lists in this case.

In this case, we're going to configure R3 to send only the route to 21.0.0.0 to R1 and 23.0.0.0 to R2. Neither R1 nor R2 will have the route to 22.0.0.0. However, we *do* want these two routers to get any *future* routes that R4 advertises into BGP.

Since these two routers will learn about these routes from an iBGP neighbor, they will not advertise the routes to each other after learning their one assigned route.

On R3, we'll write a prefix list that denies 22.0.0.0/8 and 23.0.0.0/8, but permits all other routes. This command will be applied to updates sent to R1 via the neighbor statement. After applying the command and applying a soft reset on R3, R1 sees only the 21.0.0.0 route.

```
R3(config)#ip prefix-list FILTER_R1 deny 22.0.0.0/8
R3(config)#ip prefix-list FILTER_R1 deny 23.0.0.0/8
R3(config)#ip prefix-list FILTER_R1 permit 0.0.0.0/0 le 32
R3(config)#router bgp 123
R3(config-router)#neighbor 172.12.123.1 prefix-list FILTER_R1 out
```

```
R3#clear ip bgp * soft
```

```
R1#show ip bgp
BGP table version is 6, local router ID is 19.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i21.0.0.0	172.12.123.3	0	100	0	4 i

The paths to 22.0.0.0/8 and 23.0.0.0/8 have been successfully filtered.

On R3, we'll write a prefix-list that will filter 21.0.0.0/8 and 22.0.0.0/8, but allow all other routes. After applying this prefix list to R2 via the neighbor command and performing a soft reset on R3, R2 sees only the route to 23.0.0.0.

```
R3(config)#ip prefix-list FILTER_R2 deny 21.0.0.0/8
R3(config)#ip prefix-list FILTER_R2 deny 22.0.0.0/8
R3(config)#ip prefix-list FILTER_R2 permit 0.0.0.0/0 le 32
R3(config)#router bgp 123
R3(config-router)#neighbor 172.12.123.2 prefix-list FILTER_R2 out
```

```
R3#clear ip bgp * soft
```

```
R2#show ip bgp
```

```
BGP table version is 6, local router ID is 2.2.2.2
```

```
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
```

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

Network	Next Hop	Metric	LocPrf	Weight	Path
*>i23.0.0.0	172.12.123.3	0	100	0	4 i

The paths to 21.0.0.0/8 and 22.0.0.0/8 have been successfully filtered.

To see the prefix lists configured on a route as well as the order of the statements in each list, run *show ip prefix-list*.

```
R3#show ip prefix-list
```

```
ip prefix-list FILTER_R1: 3 entries
```

```
seq 5 deny 22.0.0.0/8
```

```
seq 10 deny 23.0.0.0/8
```

```
seq 15 permit 0.0.0.0/0 le 32
```

```
ip prefix-list FILTER_R2: 3 entries
```

```
seq 5 deny 21.0.0.0/8
```

```
seq 10 deny 22.0.0.0/8
```

```
seq 15 permit 0.0.0.0/0 le 32
```

Using prefix lists properly is an important part of CCNP exam success, and for those of you with an eye on the CCIE, it's even more important. Learn as many methods as you can to filter BGP routes, and start with the fundamental method - prefix lists!

[Matching Address Classes with Prefix-Lists](#)

Instead of using ACLs to match address classes like Class A, B or C ip prefix-lists can do the trick too.

Lets first start with the class definitions:

Class A: 1.0.0.0 – 127.255.255.255

Class B: 128.0.0.0 – 191.255.255.255

Class C: 192.0.0.0 – 223.255.255.255

The reserved addresses are all in the definition above since they still belong into the definitions even though they are only used for their special function. To show how to calculate the correct ip prefix-list statement they have to be taken into the calculation.

Now if we change the class definitions from decimal into binary, the classes can also be defined with the starting bits in the first octet:

Class A: 0
Class B: 10
Class C: 110

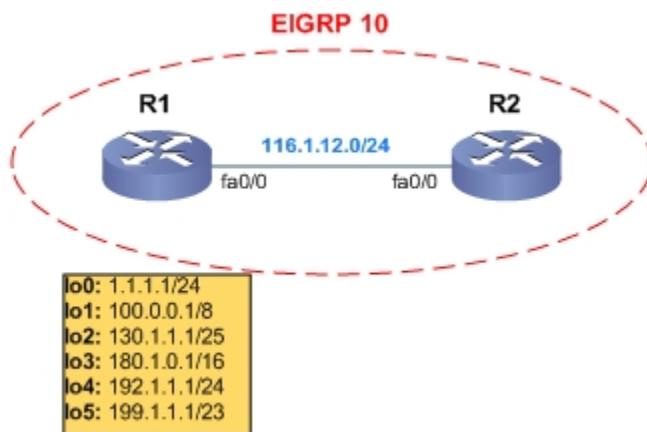
No matter what IP you take out of a class its first octet will always start in binary as shown above. Based on this list we can say that Class A needs one network bit, Class B two and Class C three to be matched on or in other words:

Class A: 1.0.0.0/1
Class B: 128.0.0.0/2
Class C: 192.0.0.0/3
Class D: 224.0.0.0/4

This is more or less how the prefix-lists can be build to match the address classes but if we use the statements above and put it together with a prefix-list, it wont work the way it should since then it would only match on the exact address and network bits. To get it working we have to expand it a bit so it matches all type of networks:

Class A: 1.0.0.0/1 le 32
Class B: 128.0.0.0/2 le 32
Class C: 192.0.0.0/3 le 32

This time the prefix-list would match on the specific classes and on all type of networks the keyword le 32 means match on all networks with a network potion of less or equal to 32 bits. To proof the working here's a little scenario with two routers and EIGRP:



R1 got six loopbacks configured, two addresses for each class. All are advertised into EIGRP and R2 got all in his routing table:

```
R2#sh ip route eigrp
1.0.0.0/24 is subnetted, 1 subnets
D 1.1.1.0 [90/156160] via 116.1.12.1, 00:00:12, FastEthernet0/0
D 100.0.0.0/8 [90/156160] via 116.1.12.1, 00:00:12, FastEthernet0/0
130.1.0.0/25 is subnetted, 1 subnets
D 130.1.1.0 [90/156160] via 116.1.12.1, 00:00:06, FastEthernet0/0
D 192.1.1.0/24 [90/156160] via 116.1.12.1, 00:00:06, FastEthernet0/0
D 180.1.0.0/16 [90/156160] via 116.1.12.1, 00:00:06, FastEthernet0/0
D 199.1.0.0/23 [90/156160] via 116.1.12.1, 00:00:06, FastEthernet0/0
```

R2 will now get three ip prefix-lists, for each class one:

```
ip prefix-list CLASS-A seq 5 permit 0.0.0.0/1 le 32
ip prefix-list CLASS-B seq 5 permit 128.0.0.0/2 le 32
ip prefix-list CLASS-C seq 5 permit 192.0.0.0/3 le 32
```

Now we can start playing around with the **distribute-list** command under the router eigrp 10

```
router eigrp 10
distribute-list prefix CLASS-A in fastEthernet 0/0
```

This will activate a filter on all ingress routing updates coming from fastethernet0/0 (or R1 in this case). You now can either wait until all routes timed out or to be faster just issue a **clear ip route ***. All routes that do not match the prefix-list CLASS-A will not be installed into the routing table so the routing table only contains the two loopback networks which belong into class A:

```
R2#sh ip route eigrp
1.0.0.0/24 is subnetted, 1 subnets
D 1.1.1.0 [90/156160] via 116.1.12.1, 00:06:44, FastEthernet0/0
D 100.0.0.0/8 [90/156160] via 116.1.12.1, 00:06:44, FastEthernet0/0
```

Changing the distribute-list from prefix-list CLASS-A to CLASS-B would result in only having the class B networks in the routing table, same with the CLASS-C prefix-list.

How to use a Distribute-List to Filter out Routing Updates in the Cisco IOS

by [David Davis, vExpert, VCP, CCIE 9369](#) - January 8, 2009

While using Dynamic routing protocols, at some point, you will want to filter the routes that are sent out from one router to another OR filter routes that are received into your router. One of the easiest ways to do this is to use a distribute-list. Let's find out how...

What is a Distribute-List?

First off, let me point out that we are not talking about a “distribution-list”. While the word “distribution” may seem to fit better, that is not what it is called. I too, over the years, have periodically called it a distribution list so I first wanted to set the record straight.

A distribute-list is used to control routing updates either coming TO your router or leaving FROM your router. Distribute-lists work on a variety of different IOS routing protocols. Because of that, learning how to use distribute-lists is very valuable.

As distribute-lists use Cisco IOS Access-Lists, you can very granularly define what routes will or won't be sent out of the router, or received into the router. Let's find out how they work...

Step 1 – Define what routes you want to filter

Let's say that you want to filter inbound routes to a router. Start off by taking a look at your current routing table. What networks, exactly, do you want to filter out? Here is a sample routing table, for our example:

Router# **show ip route**

Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP

- D - EIGRP, EX - EIGRP external, 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, E - EGP
- i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
- * - candidate default, U - per-user static route, o - ODR
- P - periodic downloaded static route

Gateway of last resort is not set

- 100.0.0.0/8 is variably subnetted, 3 subnets, 2 masks
- **O** 100.200.200.1/32 [110/11] via 172.16.100.29, 00:00:10, Ethernet0
- **O** 100.200.100.1/32 [110/11] via 172.16.100.29, 00:00:10, Ethernet0
- **C** 100.100.250.0/24 is directly connected, Loopback0
- 172.16.0.0/24 is subnetted, 1 subnets
- **C** 172.16.100.0 is directly connected, Ethernet0

Let's say that we want to filter out route 10.200.100.1/32.

Step 2 – Create an ACL to filter out that traffic

Next, we need to define an ACL that identifies that route, denies it, and allows all other traffic. Here is the ACL that I used:

```
Router(config)# access-list 50 deny 100.200.100.1
```

```
Router(config)# access-list 50 permit any
```

Step 3 – Create a Distribute-List that references the ACL and defines the direction

Now, you want to create a distribute-list that references this ACL, then specify the direction that the distribute-list will be applied.

The distribute-list is defined underneath the routing process for the protocol that it is being used on. In our case, we want to filter OSPF routes so we go into the OSPF routing process configuration.

```
Router(config)# router ospf 10
```

```
Router(config-router)#distribute-list ?
```

- <1-199> IP access list number
- <1300-2699> IP expanded access list number
- WORD Access-list name
- gateway Filtering incoming updates based on gateway
- prefix Filter prefixes in routing updates

```
Router(config-router)#distribute-list 50 ?
```

- in Filter incoming routing updates
- out Filter outgoing routing updates

```
Router(config-router)# distribute-list 50 in
```

Step 4 – Verify that the route has been removed

After you put your new ACL and distribute-list in place, verify that they were successful. Notice how, in the **show ip route** output below, the 10.200.100.1 no longer shows up.

```
Router# sh ip ro
```

```
(truncated)
```

```
100.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
```

- **O** 100.200.200.1/32 [110/11] via 172.16.100.29, 00:11:39, Ethernet0
- **C** 100.100.250.0/24 is directly connected, Loopback0
- 172.16.0.0/24 is subnetted, 1 subnets
- **C** 172.16.100.0 is directly connected, Ethernet0

Router#

Below, you will find graphics of the configurations in place on each side of this distribute-list route filtering:

```
!
interface Loopback0
 ip address 100.100.250.1 255.255.255.0
!
interface Ethernet0
 ip address 172.16.100.26 255.255.255.0
!
interface Serial0
 no ip address
!
interface Serial1
 no ip address
!
router ospf 10
 log-adjacency-changes
 network 100.0.0.0 0.255.255.255 area 0
 network 172.0.0.0 0.255.255.255 area 0
 distribute-list 50 in
!
ip classless
ip http server
ip pim bidir-enable
!
access-list 50 deny 100.200.100.1
access-list 50 permit any
!
```

Receiving Router

```
!
interface Loopback0
 ip address 100.200.100.1 255.255.255.0
!
interface Loopback1
 ip address 100.200.200.1 255.255.255.0
!
interface Ethernet0
 ip address 172.16.100.29 255.255.255.0
 no ip route-cache
!
interface Serial0
 ip address 10.1.1.1 255.255.255.0
 no keepalive
!
router ospf 10
 log-adjacency-changes
 network 100.0.0.0 0.255.255.255 area 0
 network 172.0.0.0 0.255.255.255 area 0
!
```

Sending Router

In Summary

Our route filtering with the distribute-list command was successful. You can use this same concept and procedure to filter out multiple routes from either going in or out of your router. The distribute-list feature works with a number of different routing protocols. You can even specify in the distribute-list command what interfaces you want the command applied to. So, the next time that you need to not send out a route or have a router not receive a route, don't forget about the distribute-list command (not distribution-list).

For more information on Distribute-lists, see the Cisco.com article [Filtering Routing Updates on Distance Vector IP Routing Protocols](#).

Using Extended Access-Lists In A Distribute-List

By Brian McGahan, CCIE #8593

Hi Brian,

I'm trying to create a distribute-list in RIP to allow only even routes to be received. I can do it successfully with a standard ACL, however if I use an extended ACL I can't get any routes at all. I've heard that extended ACLs are better because they also check the netmask. What am I doing wrong?

Using an extended access-list with a distribute-list is supported, however the syntax can be a little confusing because it means different things for different applications. When using an extended ACL for a distribute-list in BGP it acts like a prefix-list. This means that you can match on both the address of the prefix and the subnet mask. In other words if you have prefixes 10.0.0.0/8 and 10.0.0.0/16 you can distinguish between them by saying not only must the address be 10.0.0.0 but the subnet mask must be /8. In prefix-list syntax this is very straightforward, as to match this prefix we would use the following:

```
ip prefix-list PREFIX1 permit 10.0.0.0/8
```

When using an extended access-list in BGP the syntax of the list changes in that we are not matching source and destination pairs, but instead are matching the address and netmask. In extended ACL syntax the above prefix-list would read:

```
access-list 100 permit ip host 10.0.0.0 host 255.0.0.0
```

This means that the address must be exactly 10.0.0.0 and the subnet mask must be exactly 255.0.0.0. By changing the "host" keyword to a wildcard mask we can do fuzzy binary matches. For example the following syntax means check any address that starts with "192.168" and has a subnet mask of /24:

```
access-list 101 permit ip 192.168.0.0 0.0.255.255 host 255.255.255.0
```

In other words this list matches 192.168.0.0/24, 192.168.100.0/24, 192.168.200.0/24, etc.

This extended access-list syntax can also be used in a route-map for redistribution filtering in both IGP and BGP. For example if we took the previous access-list 101 and matched it in a route-map as follows:

```
route-map OSPF_TO_RIP permit 10
  match ip address 100
!
router rip
  redistribute ospf 1 metric 1 route-map OSPF_TO_RIP
```

This syntax would say that we want to redistribute OSPF routes into RIP, but only those which are 192.168.X.X/24.

The confusion for this extended access-list implementation is that when it is called as a distribute-list in IGP the syntax changes. In the previous examples the normal “source” field in the ACL represents the network address, where the “destination” field represents the subnet mask. In IGP distribute-list application the “source” field in the ACL matches the update source of the route, and the “destination” field represents the network address. This implementation allows us to control which networks we are receiving, but more importantly who we are receiving them from. Take the following topology:

R1, R2, and R3 share an Ethernet network 123.0.0.0/8 that is running RIP. Both R1 and R2 are advertising the identical prefixes 10.0.0.0/8 and 20.0.0.0/8 to R3. Their configurations are as follows:

```
R1#show ip int brief | exclude unassigned
```

Interface	IP-Address	OK?	Method	Status	Protocol
FastEthernet0/0	123.0.0.1	YES	manual	up	up
Loopback0	10.0.0.1	YES	manual	up	up
Loopback1	20.0.0.2	YES	manual	up	up

```
R1#show run | begin router rip
```

```
router rip
  version 2
  network 10.0.0.0
  network 20.0.0.0
  network 123.0.0.0
```

```
R2# show ip int brief | exclude unassigned
```

Interface	IP-Address	OK?	Method	Status	Protocol
FastEthernet0/0	123.0.0.2	YES	manual	up	up
Loopback0	10.0.0.1	YES	manual	up	up
Loopback1	20.0.0.2	YES	manual	up	up

```
R2#sh run | begin router rip
```

```
router rip
  version 2
  network 10.0.0.0
```

```
network 20.0.0.0
network 123.0.0.0
```

```
R3#show ip route rip
R    20.0.0.0/8 [120/1] via 123.0.0.2, 00:00:00, Ethernet0/0
      [120/1] via 123.0.0.1, 00:00:00, Ethernet0/0
R    10.0.0.0/8 [120/1] via 123.0.0.2, 00:00:00, Ethernet0/0
      [120/1] via 123.0.0.1, 00:00:00, Ethernet0/0
```

From this output we can see that R3 has the two prefixes installed twice, once from R1 and once from R2. Now let's suppose that prefix 10.0.0.0/8 we only want to receive from R1, while prefix 20.0.0.0/8 we only want to receive from R2. We can accomplish this with an extended access-list as follows:

```
R3#conf t
Enter configuration commands, one per line.  End with CNTL/Z.

R3(config)#access-list 100 permit ip host 123.0.0.1 host 10.0.0.0
R3(config)#access-list 100 permit ip host 123.0.0.2 host 20.0.0.0
R3(config)#router rip
R3(config-router)#distribute-list 100 in Ethernet0/0
R3(config-router)#end
R3#clear ip route *
R3#show ip route rip
R    20.0.0.0/8 [120/1] via 123.0.0.2, 00:00:00, Ethernet0/0
R    10.0.0.0/8 [120/1] via 123.0.0.1, 00:00:00, Ethernet0/0
```

We can see now R3 only has one entry for each prefix, with the 10.0.0.0/8 coming only from R1 and the 20.0.0.0/8 coming only from R2. The disadvantage of this application however is that we cannot distinguish prefixes based on their netmask. For example we could not say that we want to receive prefix 172.16.0.0/16 from only R1 and prefix 172.16.0.0/24 only from R2. For this implementation in IGP we would use a prefix-list that is called from a distribute-list with the "distribute-list prefix" syntax under the routing process.