

Configuring Routing Policy

1

This chapter provides information on configuring routing policy for your ERX system. It describes routing policy configuration in general as it might be used with various routing protocols, such as BGP, IS-IS, OSPF, and RIP.

Topic	Page
Overview	1-1
References	1-2
Route Maps	1-2
Access Lists	1-15
Using the Null Interface	1-24
Prefix Lists	1-24
Prefix Trees	1-26
Community Lists	1-28
Using Regular Expressions	1-33
Managing the Routing Table	1-39
Troubleshooting Routing Policy	1-40
Monitoring Routing Policy	1-40

Overview

Routing policy determines how the system handles the routes it receives from and sends to neighboring routers. In many cases, routing policy consists of filtering routes, accepting certain routes, accepting and modifying other routes, rejecting some routes, and determining the

routing protocol used to distribute the routes. You can think of routing policy as a way to control the flow of routes into and out of the system.

The decision about which routes to accept from and advertise to various neighbors has an important impact on the traffic that crosses a network. Routing policy is used to enforce business agreements between two or more ISPs concerning the amount and type of traffic that is allowed to pass between them.

You can use one or more of the following mechanisms to configure routing policy:

- Route maps
- Access lists
- Prefix lists
- Prefix trees
- Community lists

References

For more information about the protocols discussed in this chapter, refer to their respective chapters in this guide and in *ERX Routing Protocols Configuration Guide, Vol. 2*, and to the References sections within those chapters.

Route Maps

You can use route maps to control and modify routing information and to define conditions for redistributing routes between routing domains. You can apply route maps to inbound, outbound, or redistribution routes. A route map consists of *match* clauses and *set* clauses.

Match clauses specify the attribute values that determine whether a route matches the route map. A route that has the same attribute values passes the match condition. Routes that pass all the match conditions match the route map. You issue **match** commands to define the match conditions for a route map. You can specify the match conditions in any order. If you do not specify *any* match conditions in a route map, that route map then matches *all* routes.

Set clauses define how the attributes are modified for matching routes. The set conditions apply only to routes that pass all the match conditions (or a route map with no match conditions). When a route passes all the

match conditions, all set conditions are applied. You issue **set** commands to define the set conditions for a route map.

You assign a unique alphanumeric string called the *map tag* to identify each route map. You can have multiple *instances* of a given route map, where each instance consists of a different group of clauses. Each instance is identified by a *sequence number*. When you apply a route map, the routing protocol evaluates routes against the instance of the route map having the lowest sequence number. If the routes pass all the match conditions specified in the lowest-numbered instance, and if all **set** commands are successfully applied, then no other instance of the route map is considered. However, any routes that do not pass all the match conditions are evaluated against the next instance of the route map. For example, suppose you create two instances of route map boston5, one with sequence number 10 and one with sequence number 25. When you apply boston5, routes are evaluated first against instance 10; any that do not match are evaluated against instance 25.

When you apply a route map, you specify the **permit** or **deny** keyword. If you specify the **permit** keyword, routes that match the route map are accepted, forwarded, or redistributed. Routes that do not match the route map are rejected or blocked.

If you specify the **deny** keyword, routes that match the route map are rejected or blocked. Routes that do not match the route map are accepted, forwarded, or redistributed.

A route map must have at least one match clause or one set clause. If you have no match clauses, all routes match the route map, and the set conditions apply to all routes. If you have no set clauses, no action is taken other than that specified by the **permit** or **deny** keyword.

Sample Route Map

Consider the network structure shown in Figure 1-1. Suppose you don't want router Boston to receive any routes that originate in or pass through router Chicago.

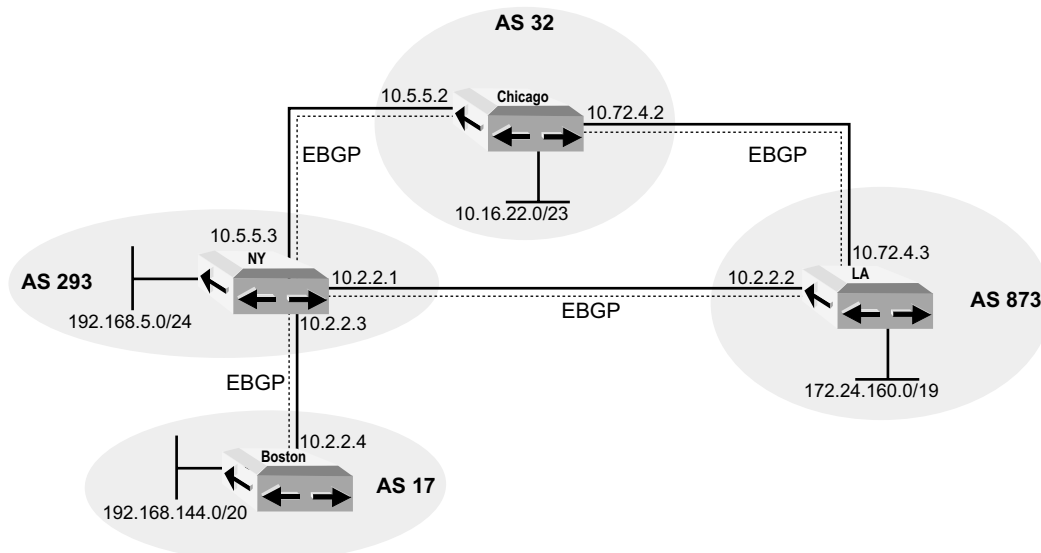


Figure 1-1 Applying route maps to routes

You can use a route map to filter routes based on the AS path to accomplish this goal. Use the following commands to configure router NY:

```

host1(config)#router bgp 293
host1(config-router)#network 192.168.5.0 mask 255.255.255.0
host1(config-router)#neighbor 10.5.5.2 remote-as 32
host1(config-router)#neighbor 10.2.2.2 remote-as 873
host1(config-router)#neighbor 10.2.2.4 remote-as 17
host1(config-router)#neighbor 10.2.2.4 route-map block1 out
host1(config-router)#exit
host1(config)#ip as-path access-list boston deny _32_
host1(config)#route-map block1 deny 1
host1(config-route-map)#match as-path boston

```

Multiple Values in a Match Entry

You can specify more than one value in each match entry of a route map configured by any of the **match** commands shown in Table 1-1. A clause with multiple values matches a route having any of the values; that is, the multiple values are logical ORed.

Table 1-1 Match commands enabling multiple values

match as-path	match ip next-hop
match community	match level
match distance	match metric
match extcommunity-list	match route-type
match ip address	match tag

```
host1(config-route-map)#match ip address lisbon madrid  
host1(config-route-map)#match as-path 10 20 30
```

You can also issue successive **match** commands to add new values to a route map entry for any of the commands in Table 1-1.

```
host1(config-route-map)#match ip address boston  
host1(config-route-map)#match ip address newyork
```

This is equivalent to issuing the following single command:

```
host1(config-route-map)#match ip address boston newyork
```

You cannot specify multiple values for the **match metric-type** command, because it has only two acceptable values, which are mutually exclusive. Specifying both values would have the same effect as not specifying a metric type at all; specifying the same value more than once has no meaning.

Negating Match Clauses

If you specify a value when you negate a **match** command configured in a route map, only that value for the match entry is deleted. The entire match entry is deleted only if no other values are specified by the entry. In earlier releases, any value specified with a **no match** command was ignored, and the entire match entry was deleted. This change applies to all **match** commands configured in a route map.

For example, consider the following match entry to route map miami:

```
host1(config)#ip community-list corporate5 permit 32 463 21  
host1(config)#ip community-list dade2 permit 41 53 22  
host1(config)#route-map miami permit 1  
host1(config-route-map)#match community corporate5 dade2
```

```

host1(config-route-map)#exit
host1(config)#exit
host1#show route-map
route-map miami, permit, sequence 10
  Match clauses:
    match community corporate5 dade2

```

In earlier releases, issuing a command to remove a community not specified in the entry deleted the whole entry, but now nothing happens:

```

host1(config-route-map)#no match community southbeach
host1(config-route-map)#exit
host1(config)#exit
host1#show route-map
route-map miami, permit, sequence 10
  Match clauses:
    match community corporate5 dade2

```

If you instead issue the following commands, the specified value is deleted:

```

host1(config-route-map)#no match community dade2
host1(config-route-map)#exit
host1(config)#exit
host1#show route-map
route-map miami, permit, sequence 10
  Match clauses:
    match community corporate5

```

You could delete the entire match community entry by issuing either of the following commands:

```

host1(config-route-map)#no match community
host1(config-route-map)#no match community corporate5 dade2

```

Matching a Community List Exactly

You can use the **exact-match** keyword for the **match community** command to specify that a match exists only for the exact community numbers specified in the community list. The **exact-match** keyword applies only to a standard community list—that is, one not specified by a regular expression. You cannot use the **exact-match** keyword with a community list that is specified by a regular expression.

Consider the following example:

```

host1(config)#ip community-list 1 permit 100 200 300
host1(config)#exit
host1#show ip community-list
Community standard list 1

```

```

    permit 0:100 0:200 0:300
host1(config)#route-map example1 permit 10
host1(config-route-map)#match community 1 exact-match
host1(config)#exit
host1#show route-map example1
route-map example, permit, sequence 10
  Match clauses:
    community (community-list filter): 1 exact-match

```

The route map *example1* permits a route only if the route contains community 100 and community 200 and community 300 and no additional communities.

If you do not specify the **exact-match** option, the route map also permits a match on a route containing additional communities. For example, a route containing communities 100, 200, 300, 400, and 450 would match.

Removing Community Lists from a Route Map

You can use the **set comm-list delete** command to remove the specified community list from routes matching the route map, provided that you created the community list with a single community number per list entry. For example, you cannot remove the community lists 231:10 and 231:20 with the **set comm-list delete** command if you created them with the following command:

```

host1(config)#ip community list 1 permit 231:10 231:20

```

You can, however, remove the lists with the **set comm-list delete** command if you created them separately:

```

host1(config)#ip community list 1 permit 231:10
host1(config)#ip community list 1 permit 231:20

```

Redistributing Access Routes

The following example shows how to redistribute access and access-internal routes by matching on a tag.

- 1 Configure route map 2 to match tag 30:

```

host1(config)#route-map 2
host1(config-route-map)#match tag 30

```

- 2 Configure redistribution into BGP of the access routes with route map 1:

```

host1(config)#router bgp testnet
host1(config-router)#redistribute access

```

match as-path

- Use to match an AS-path access list.
- The implemented weight is based on the first matched AS path.
- Example

```
host1(config-route-map)#match as-path pathlist5
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match community

- Use to match a community list.
- Supported for inbound and outbound route maps.
- Example

```
host1(config-route-map)#match community comm5
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match distance

- Use to match any routes being redistributed out of the routing table that have the specified administrative distance.
- Distance is used to determine the relative preference between routes to the same prefix in order to pick the best route to that prefix in the routing table. Distance has no meaning in any other circumstance, and any attempt to match distance will fail.
- Example

```
host1(config-route-map)#match distance 25
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match extcommunity

- Use to match an extended community list in a route map.
- You can specify one or more extended community list names in a match clause. If you specify more than one extended community list, the lists are logically ORed.
- Example

```
host1(config-route-map)#match extcommunity topeka10
```
- Use the **no** version to remove the match clause from a route map or a specified value from the match clause.

match ip address

- Use to match any route that has a destination network number that is permitted by an access list, a prefix list, or a prefix tree, or performs policy routing on packets.
- Example

```
host1(config-route-map)#match ip address prefix-tree boston
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match ip next-hop

- Use to match any routes that have a next-hop router address passed by the specified access list, prefix list, or prefix tree.
- Example

```
host1(config-route-map)#match ip next-hop 5 192.54.24.1
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match level

- Use to match routes for the specified type.
- Example

```
host1(config-route-map)#match level-1
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match metric

- Use to match a route for the specified metric value.
- Example

```
host1(config-route-map)#match metric 10
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match metric-type

- Use to match a route for the specified metric type.
- Example

```
host1(config-route-map)#match metric-type external
```
- Use the **no** version to delete the match clause from a route map.

match route-type

- Use to match a route for the specified route type.
- Example

```
host1(config-route-map)#match route-type level-1
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match-set summary prefix-tree

- Use to specify the prefix tree that summarizes routes for a particular route map.
- Use the **ip prefix-tree** command to set the conditions of the prefix tree, including which routes to summarize and how many bits of the network address to preserve.
- Example

```
host1(config-route-map)#match-set summary prefix-tree boston
```
- Use the **no** version to disable the use of the prefix tree by the route map.

match tag

- Use to match the tag value of the destination routing protocol.
- Example

```
host1(config)#route-map 1  
host1(config-route-map)#match tag 25
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

route-map

- Use to define the conditions for redistributing routes from one routing protocol into another, and for filtering or modifying updates sent to or received from peers.
- Each **route-map** command has a list of **match** and **set** commands associated with it. That is, the route map itself consists of a set of clauses; each clause (also called an entry) consists of a **match** or **set** command.
 - **match** commands specify the match criteria, the conditions under which redistribution is allowed for the current route map.
 - **set** commands specify the set actions, the redistribution actions to perform if the criteria enforced by the **match** commands are set.
- You can specify match and set clauses to modify attributes of redistributed routes.
- Use route maps when you wish to have detailed control over how routes are redistributed between routing processes.
 - › You specify the destination routing protocol with the **router** command.
 - › You specify the source routing protocol with the **redistribute** command.
- Example

```
host1(config)#route-map nycl permit 10
```

```
host1(config-route-map)#match ip address list1  
host1(config-route-map)#set metric-type internal
```

- Use the **no** version to delete the route map.

set as-path prepend

- Use to modify an AS path for BGP routes by prepending one or more AS numbers or a list of AS numbers to the path list.
- The only global BGP metric available to influence the best path selection is the AS path length. By varying the length of the AS path, a BGP speaker can influence the best path selection by a peer farther away.

- Example

```
host1(config-route-map)#set as-path prepend list list10
```

- Use the **no** version to delete the set clause from a route map.

set automatic-tag

- Use to automatically compute the tag value of the destination routing protocol.

- Example

```
host1(config-route-map)#set automatic-tag
```

- Use the **no** version to delete the set clause from a route map.

set comm-list delete

- Use to remove communities specified by the community list from the community attribute of routes matching the route map.
- You can use this command to delete communities only if the community list was created with a single community per list entry, as shown in the following sample configuration for router Test:

```
host1(config)#ip community-list 1 permit 231:10  
host1(config)#ip community-list 1 permit 231:20  
host1(config)#router bgp 45  
host1(config-router)#neighbor 10.6.2.5 remote-as 5  
host1(config-router)#neighbor 10.6.2.5 route-map indelete in  
host1(config-router)#route-map indelete permit 10  
host1(config-route-map)#set comm-list 1 delete
```

Router Test receives the same route from 10.6.2.5 and applies the indelete route map. BGP compares each list entry with the community attribute. A match is found for the list entry 231:10, and this community is deleted from the community attribute. Similarly, a match is found for the list entry of 231:20, and this community is deleted from the community attribute.

- Use the **no** version to delete the set clause from a route map.

set community

- Use to set the community attribute in BGP updates.
- You can specify a community list number in the range 1–4294967295, or in the new community format of AA:NN, or one of the following well-known communities:
 - › **local-as** – prevents advertisement outside of the local AS
 - › **no-advertise** – prevents advertisement to any peer
 - › **no-export** – prevents advertisement beyond the BGP confederation boundary
- Alternatively, you can use the **list** keyword to specify the name of a community list that you previously created with the **ip community-list** command.
- Example

```
host1(config-route-map)#set community no-advertise
```
- Use the **none** keyword to remove the community attribute from a route.
- Use the **no** version to delete the set clause from a route map.

set dampening

- Use to enable BGP route flap dampening only on routes that pass the match clauses of, and are redistributed by, a particular route map.
- BGP creates a dampening parameter block for each unique set of dampening parameters—such as suppress threshold, reuse threshold, and so on—used by BGP. For example, if you have a route map that sets the dampening parameters to one set of values for some routes and to another set of values for the remaining routes, BGP uses and stores two dampening parameter blocks, one for each set.
- Example

```
host1(config-route-map)#set dampening 5 1000 1500 45 15
```
- Use the **no** version to delete the set clause from a route map.

set distance

- Use to set the administrative distance attribute on routes being installed into the routing table that match the route map.
- Distance is used to establish preference between routes to the same prefix to identify the best route to that prefix. Setting distance in any other circumstance has no effect.
- Example

```
host1(config-route-map)#set distance 5
```
- Use the **no** version to delete the set clause from a route map.

set extcommunity

- Use to set the extended community attributes in a route map for BGP updates.
- You can specify a site-of-origin (**soo**) extended community and a route target (**rt**) extended community at the same time in a set clause without overwriting the other.
- Example

```
host1(config-route-map)#set extcommunity rt 10.10.10.2:325
```
- Use the **no** version to delete the set clause from a route map.

set ip next-hop

- Use to set the next hop attribute of a route that matches a route map.
- You can specify an IP address or an interface as the next hop.
- Use the peer-address keyword to have the following effect:
 - › On outbound route maps, disables the next hop calculation by setting the next hop to the IP address of the BGP speaker
 - › On inbound route maps, overrides any third-party next hop configuration by setting the next hop to the IP address of the peer
- Example

```
host1(config-route-map)#set ip next-hop 192.56.32.1
```
- Use the **no** version to delete the set clause from a route map.

set level

- Use to specify where to import routes when all of a route map's match criteria are met.
- Example

```
host1(config-route-map)#set level level-2
```
- Use the **no** version to delete the set clause from a route map.

set local-preference

- Use to specify a preference value for the AS path.
- Example

```
host1(config-route-map)#set local-preference 200
```
- Use the **no** version to delete the set clause from a route map.

set metric

- Use to set the metric value—for BGP, the MED—for a route.
- To establish an absolute metric, do not enter a plus or minus sign before the metric value.
- Example

```
host1(config-route-map)#set metric 10
```

- To establish a relative metric, specify a plus or minus sign immediately preceding the metric value. The value is added to or subtracted from the metric of any routes matching the route map. The relative metric value can range from 0 to 4294967295.
- Example


```
host1(config-route-map)#set metric -25
```
- You cannot use both an absolute metric and a relative metric within the same route map sequence. Setting either metric overrides any previously configured value.
- Use the **no** version to delete the set clause from a route map.

set metric-type

- Use to set the metric type for a route.
- For BGP, this command affects BGP behavior only in outbound route maps and has no effect on other types of route maps. If the route map contains both a **set metric-type** and a **set metric** clause, the **set metric** clause takes precedence. If you specify the **internal** metric type in a BGP outbound route map, BGP sets the MED of the advertised routes to the IGP cost of the next hop of the advertised route. If the cost of the next hop changes, BGP is not forced to readvertise the route.
- For BGP, you can specify the following:
 - › **external** – reverts to the normal BGP rules for propagating the MED; this is the BGP default
 - › **internal** – sets the MED of a received route that is being propagated to an external peer equal to the IGP cost of the indirect next hop
- For IS-IS, you can specify the following:
 - › **external** – only the metric of the route itself is considered for comparison
 - › **internal** – both the metric of the route and the cost to the router that advertised the route are considered for comparison; this is the IS-IS default
- For OSPF, you can specify the following:
 - › **1** – cost of the external routes is equal to the sum of all internal costs and the external cost
 - › **2** – cost of the external routes is equal to the external cost alone; this is the OSPF default
- Example


```
host1(config-route-map)#set metric-type internal
```
- Use the **no** version to delete the set clause from a route map.

set origin

- Use to set the BGP origin of the advertised route.
- Example


```
host1(config-route-map)#set origin egp
```
- Use the **no** version to delete the set clause from a route map.

set route-type

- Use to set the routes of the specified type.
- Example

```
host1(config-route-map)#set route-type internal
```
- Use the **no** version to delete the set clause from a route map.

set tag

- Use to set the tag value of the destination routing protocol.
- Example

```
host1(config-route-map)#set tag 23
```
- Use the **no** version to delete the set clause from a route map.

set weight

- Use to specify the BGP weight for the routing table.
- The weights assigned with the **set weight** command in a route map override the weights assigned using the **neighbor weight** and **neighbor filter-list weight** commands.
- Example

```
host1(config-route-map)#set weight 200
```
- Use the **no** version to delete the set clause from a route map.

Access Lists

An access list is a sequential collection of permit and deny conditions that you can use to filter inbound or outbound routes. You can use different kinds of access lists to filter routes based on either the prefix or the AS path.

Filtering Prefixes

To filter routes based on the prefix, you can do any of the following:

- Define an access list with the **access list** command, and apply the list to routes received from or passed to a neighbor with the **neighbor distribute-list** command
- Define a prefix list with the **ip prefix-list** command, and apply the list to routes received from or passed to a neighbor with the **neighbor prefix-list** command
- Define a prefix tree with the **ip prefix-tree** command, and apply the list to routes received from or passed to a neighbor with the **neighbor prefix-tree** command

The router compares each route's prefix against the conditions in the list or tree one by one. If the first match is for a permit condition, the route is accepted or passed. If the first match is for a deny condition, the route is rejected or blocked. The order of conditions is critical because testing stops with the first match. If no conditions match, the router rejects or blocks the address; that is, the last action of any list is an implicit deny condition for all routes. The implicit rule is displayed by **show access-list** and **show config** commands.

You cannot selectively place conditions in or remove conditions from an access list, prefix list, or prefix tree. You can insert a new condition only at the end of a list or tree.

Example 1 The following example shows the how the implicit deny condition is displayed.

```
host1(config)#access-list 1 permit 10.10.10.1 0.0.0.255
host1(config)#access-list 2 permit 10.25.25.1 0.0.0.255
host1(config)#access-list 3 permit any any
host1(config)#show access-list
IP Access List 1:
  permit ip 10.10.10.1 0.0.0.255 any
  deny ip any any
IP Access List 2:
  permit ip 10.25.25.1 0.0.0.255 any
  deny ip any any
IP Access List 3:
  permit ip any any
```

Note that the implicit deny rule does not appear in the display for access list 3, because any prefix will match access list 3.

Example 2 The following example demonstrates how to use a route map and an access list to redistribute static routes to IS-IS.

1 Configure 3 static routes:

```
host1(config)#ip route 20.20.20.0 255.255.255.0 192.168.1.0
host1(config)#ip route 20.20.21.0 255.255.255.0 192.168.2.0
host1(config)#ip route 20.21.0.0 255.255.255.0 192.168.30.0
```

2 Configure an access list, fltra, that filters routes 20.20.20.0/24 and 20.20.21.0/24:

```
host1(config)#access-list fltra permit 20.20.0.0 0.0.255.255
```

3 Configure route map 1 to match the access list fltra, and apply an internal metric type:

```
host1(config)#route-map 1
```

```
host1(config-route-map)#match ip address fltra
host1(config-route-map)#set metric-type internal
```

- 4 Configure redistribution into IS-IS of the static routes with route map 1:

```
host1(config)#router isis testnet
host1(config-router)#redistribute static route-map 1
```

- 5 Use the **show isis database** command to verify the effect of the redistribution (the two static routes matching the route map are redistributed as level 2 internal routes):

```
host1#show isis database detail l2
IS-IS Level-2 Link State Database
LSPID LSP Seq Num LSP Checksum LSP Holdtime ATT/P/OL
0000.0000.6666.00-00 0x000002B7 0x3E1F 1198 0/0/0
Area Address: 47.0005.80FF.F800.0000.0001.0001
NLPID: 0xcc
IP Address: 192.168.1.105
Metric: 10 IS 0000.0000.6666.01
Metric: 10 IS 0000.0000.3333.00
Metric: 10 IS 0000.0000.7777.00
Metric: 30 IP 20.20.20.0 255.255.255.0
Metric: 30 IP 20.20.21.0 255.255.255.0
```

Example 3 The following example demonstrates how to use an access list to filter routes advertised to a BGP speaker. Consider the network structure in Figure 1-2.

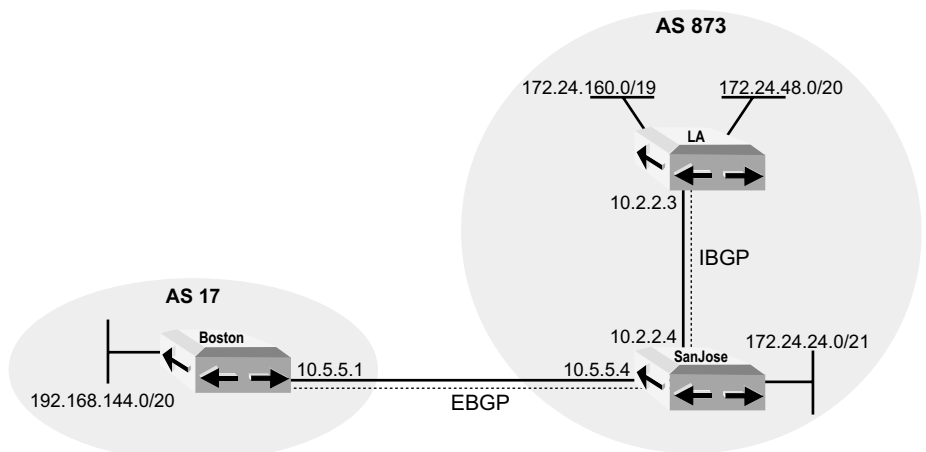


Figure 1-2 Filtering with access lists

The following commands configure router Boston to apply access list `reject1` to routes inbound from router SanJose. Access list `reject1` rejects routes matching `172.24.160.0/19`.

```
host1(config)#router bgp 17
host1(config-router)#neighbor 10.5.5.4 remote-as 873
host1(config-router)#neighbor 10.5.5.4 distribute-list
    reject1 in
host1(config-router)#exit
host1(config)#access-list reject1 permit 172.24.48.0 0.0.255
host1(config)#access-list reject1 deny 172.24.160.0
    0.0.0.255
host1(config)#access-list reject1 permit 172.24.24.0
    0.0.0.255
```

Filtering AS Paths

You can use a filter list to filter incoming and outgoing routes based on the value of the AS-path attribute. Whenever a BGP route passes through an AS, BGP prepends its AS number to the AS-path attribute. The AS-path attribute is the list of ASs that a route has passed through to reach a destination.

To filter routes based on the AS path, define the access list with the **ip as-path access-list** command, and apply the list to routes received from or passed to a neighbor with the **neighbor filter-list** command. AS-path access lists use regular expressions to describe the AS path to be matched. A regular expression uses special characters—often referred to as metacharacters—to define a pattern that is compared with an input string. For a full discussion of regular expressions, with examples on how to use them, see *Using Regular Expressions* later in this chapter.

The router compares each route's AS path against each condition in the access list. If the first match is for a permit condition, the route is accepted or passed. If the first match is for a deny condition, the route is rejected or blocked. The order of conditions is critical because testing stops with the first match. If no conditions match, the router rejects or blocks the route; that is, the last action of any list is an implicit deny condition for all routes.

You cannot selectively place conditions in or remove conditions from an AS-path access list. You can insert a new condition only at the end of an AS-path access list.

Example 1 Consider the network structure in Figure 1-3.

Suppose you want router London to behave in the following way:

- Accept routes originated in AS 621 only if they pass directly to router London.
- Accept routes originated in AS 11 only if they pass directly to router London.
- Forward routes from AS 282 to AS 435 only if they pass through either AS 621 or AS 11, but not both AS 621 and AS 11.

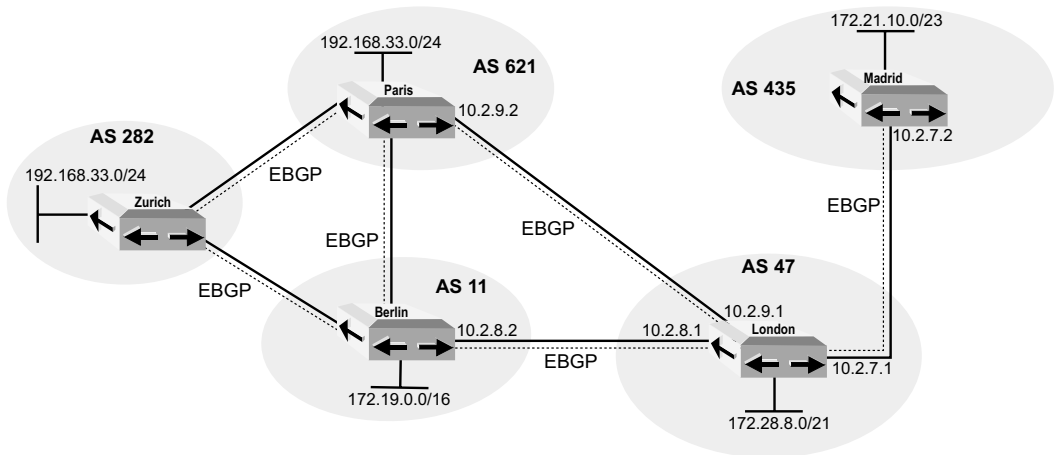


Figure 1-3 Filtering with AS-path access lists

The following commands configure router London to apply filters based on AS path to routes received from router Berlin and router Paris and to routes forwarded to router Madrid.

```

host1(config)#router bgp 47
host1(config-router)#neighbor 10.2.9.2 remote-as 621
host1(config-router)#neighbor 10.2.9.2 filter-list 1 in
host1(config-router)#neighbor 10.2.8.2 remote-as 11
host1(config-router)#neighbor 10.2.8.2 filter-list 2 in
host1(config-router)#neighbor 10.2.7.2 remote-as 435
host1(config-router)#neighbor 10.2.7.2 filter-list 3 out
host1(config-router)#exit
host1(config)#ip as-path access-list 1 deny ^11
host1(config)#ip as-path access-list 1 permit .*
host1(config)#ip as-path access-list 2 deny ^621
host1(config)#ip as-path access-list 2 permit .*
host1(config)#ip as-path access-list 3 deny [621 11]
host1(config)#ip as-path access-list 3 permit .*
    
```

AS-path access list 1 is applied to routes that router London receives from router Paris. Router London rejects routes with the AS path 11 621 or 11 282 621.

AS-path access list 2 is applied to routes that router London receives from router Berlin. Router London rejects routes with the AS path 621 11 or 621 282 11.

Router London accepts routes with the AS path 282 11, 282 621, 282 621 11, or 282 11 621. However, it applies AS-path access-list 3 to routes it forwards to router Madrid, and filters out routes with the AS path 282 621 11 or 282 11 621.

Using Access Lists in a Route Map

You can use a route map instead of the **neighbor filter-list** command to apply access lists for filtering routes. In Figure 1-4, a route map is used to determine the weight for routes learned by router Chicago.

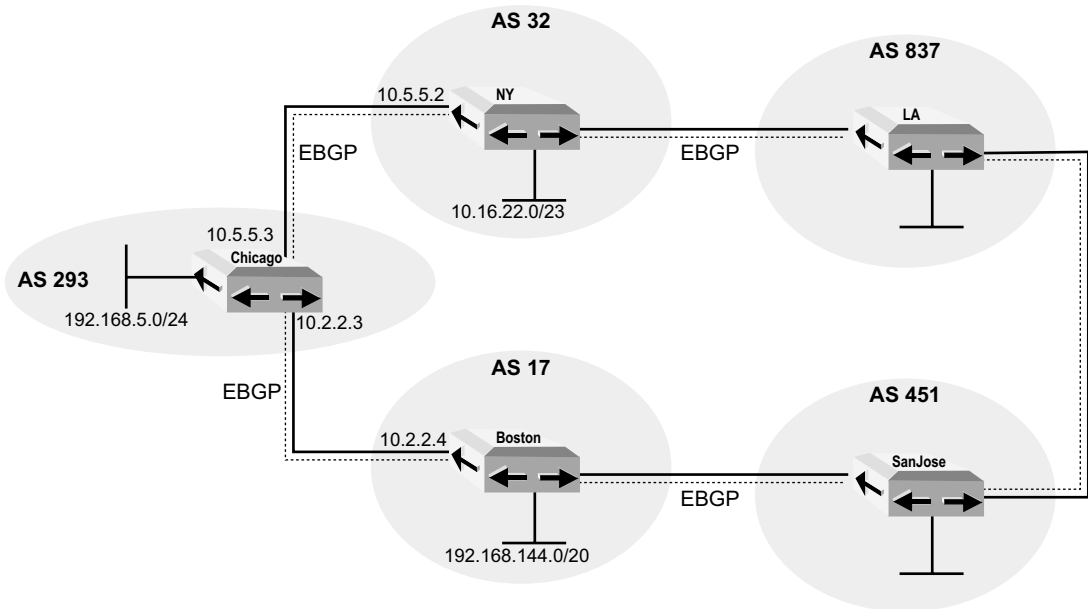


Figure 1-4 Route map filtering

Access list 1 permits any route whose AS-path attribute includes 32 or 837. This condition permits routes that originate in (or pass through from elsewhere) AS 32 or AS 837. When these routes are advertised through AS 451 and AS 17 to router Chicago, instance 1 of route map 1 matches such routes and sets their weight to 25, overriding the neighbor weight set for updates received from 10.2.2.4.

Access list 2 permits any route whose AS-path attribute indicates that it originates in AS 74. When these routes are advertised through AS 837 and AS 32 to router Chicago, instance 1 of route map 2 matches such routes and sets their weight to 175, overriding the neighbor weight set for updates received from 10.5.5.2.

To configure router Chicago:

```
host1(config)#router bgp 293
host1(config-router)#network 192.168.5.0 mask 255.255.255.0
host1(config-router)#neighbor 10.2.2.4 remote-as 17
host1(config-router)#neighbor 10.2.2.4 weight 150
host1(config-router)#neighbor 10.2.2.4 route-map 1 in
host1(config-router)#exit

host1(config-router)#neighbor 10.5.5.2 remote-as 32
host1(config-router)#neighbor 10.5.5.2 weight 50
host1(config-router)#neighbor 10.5.5.2 route-map 2 in

host1(config)#route-map 1 permit 1
host1(config-route-map)#match as-path 1
host1(config-route-map)#set weight 25
host1(config-route-map)#exit
host1(config)#ip as-path access-list 1 permit [ 32 837 ]

host1(config)#route-map 2 permit 1
host1(config-route-map)#match as-path 2
host1(config-route-map)#set weight 175
host1(config-route-map)#exit
host1(config)#ip as-path access-list 2 permit [ 74 ]
```

The result of this configuration is that router Chicago prefers routes learned via router Boston (weight 150) over routes learned through router NY (weight 50), except that:

- Router Chicago prefers routes learned via router NY that passed through AS 837 or AS 32 (weight 50) over the same routes learned via router Boston (weight 25 per route map 1).
- Router Chicago prefers routes originating in AS 74 learned via router NY that passed through AS 837 and AS 32 (weight 175 per route map 2) over the same routes learned via router Boston (weight 150).

access-list

- Use to define an IP access list to permit or deny routes based on the prefix.
- Each access list is a set of permit or deny conditions for routes based on matching a route's prefix.
- A zero in the wildcard mask means that the corresponding bit in the address must be exactly matched by the route. A one in the wildcard mask means that the corresponding bit in the address does not have to be matched by the route.
- Use the **neighbor distribute-list** command to apply the access list to routes received from or forwarded to a neighbor.
- Use the **log** keyword to log an Info event in the ipAccessList log whenever an access-list rule is matched.
- Use the **no** version to delete an IP access list (no other options specified), the specified entry in the access list, or the log for the specified access list or entry (by specifying the **log** keyword).

default-information originate

- Use to enable RIP, OSPF, or BGP to advertise a default route (0.0.0.0/0) that exists in the IP routing table.
- If you specify the **always** option for OSPF, OSPF generates a default route if it does not exist in the IP routing table and advertises it.
- Use to generate a default route into an IS-IS domain.
- Example

```
host1(config-router)#default-information originate
```
- Use the **no** version to disable advertisement of the default route.

ip as-path access-list

- Use to define an AS-path access list to permit or deny routes based on the AS path.
- Each access list is a set of permit or deny conditions for routes based on matching a route's AS path to a regular expression. If the regular expression matches the representation of the AS path of the route as an ASCII string, then the permit or deny condition applies. The AS path does not contain the local AS number.
- The AS path allows substring matching. For example, the regular expression *20* matches AS path = *20* and AS path = *100 200 300*, because *20* is a substring of each path. To disable substring matching and constrain matching to only the specified attribute string, place the underscore (*_*) metacharacter on both sides of the string; for example *_20_*.
- Use the **neighbor filter-list** command to apply the AS-path access list. You can apply access list filters to inbound and outbound BGP routes. You can assign weights to routes matching the AS-path access list.
- Use the **no** version to remove the AS-path access list; all entries belonging to this list are removed.

neighbor distribute-list

- Use to filter routes to selected prefixes as specified in an access list. Distribute lists are applied only to external peers.
- Use the **in** keyword to apply the list to inbound routes (inbound policy). Use the **out** keyword to apply the list to outbound routes (outbound policy).
- Using distribute lists is one of several ways to filter BGP advertisements. Other methods are to:
 - › Use AS path filters with the **ip as-path access-list** and the **neighbor filter-list** commands
 - › Use route map filters with the **route-map** and the **neighbor route-map** commands
- Use the **no** version to disassociate the access list from a neighbor.

neighbor filter-list

- Use to assign an AS-path access list to matching inbound or outbound routes.
- Use the **in** keyword to apply the list to inbound routes (inbound policy). Use the **out** keyword to apply the list to outbound routes (outbound policy).
- You can specify an optional weight value with the **weight** keyword to assign a relative importance to incoming routes matching the AS-path access list.
- Access list values can range from 0–65535.
- Use the **no** version to disassociate the access list from a neighbor.

neighbor prefix-list

- Use to assign an inbound or outbound prefix list.
- If you specify a BGP peer group by using the *peer-group-name* argument, all the members of the peer group inherit the characteristic configured with this command unless it is overridden for a specific peer.
- Example

```
host1(config-router)#neighbor 192.168.1.158 prefix-list
seoul19 in
```
- Use the **no** version to remove the prefix list.

neighbor prefix-tree

- Use to assign an inbound or outbound prefix tree.
- If you specify a BGP peer group by using the *peer-group-name* argument, all the members of the peer group inherit the characteristic configured with this command unless it is overridden for a specific peer.
- Example

```
host1(config-router)#neighbor 192.168.1.158 prefix-tree
newyork out
```
- Use the **no** version to remove the prefix tree.

redistribute

- Use to redistribute routes from one routing domain into another routing domain.
- Example

```
host1(config)#router bgp 100
host1(config-router)#neighbor 192.56.10.2 remote-as 200
host1(config-router)#redistribute static
host1(config-router)#exit
host1(config)#ip route 155.30.0.0 0.0.255.255
```

- Use the **no** version to remove the filter list.

Using the Null Interface

You can use access control lists to filter undesired traffic. Another way to handle undesired traffic is to send it to the null interface. The null interface is always up and acts as a data sink. The null interface cannot forward or receive traffic. The null interface is created by default; you do not have to create it manually. Direct traffic to the null interface by specifying the **null 0** keywords instead of a next-hop or destination address when configuring routes.

ip route

- Use to configure a static route and redirect traffic from it to the null interface.
- Example

```
host1(config)#ip route 10.10.20.5 null 0
```

- Use the **no** version to remove the static route.

Prefix Lists

A prefix list is a sequential collection of permit and deny conditions that apply to IP addresses. Like an access list, the system tests addresses one by one against the conditions in a prefix list. The first match determines whether the system accepts or rejects the address. Because the system stops testing conditions after the first match, the order of the conditions is critical. If no conditions match, the system rejects the address. An empty prefix list results in an automatic permit of the tested address.

Unlike access lists, the prefix list specifies a base IP address and a length, the number of bits applied to the base to determine the network prefix. The tested address is matched against the prefix.

Use the **ip prefix-list** command to define an IP prefix list. Use the **prefix-list** keyword with the **match ip address** or **match ip next-hop** commands to add a clause to a route map.

Using a Prefix List

The following example creates a prefix list that permits routes with a prefix length up to 24 in the 151.0.0.0/8 network:

```
host1(config)#ip prefix-list abc permit 151.0.0.0/8 le 24
```

clear ip prefix-list

- Use to clear all hit counts in the prefix lists, or the specified entry from the specified prefix list. (The hit count is incremented +1 each time an entry matches.)

- Example

```
host1#clear ip prefix-list abc
```

- There is no **no** version.

ip prefix-list

- Use to create a prefix list for route filtering and to specify a list entry—a deny or permit clause for a network address—to the prefix list. Use to add entries to prefix lists.
- The prefix list name can be up to 32 characters long.
- Specify the position of each entry in the list with the **seq** keyword. If a sequence number is not specified, the value of the last sequence number plus 5 is used.
- Use the **ge** and **le** keywords to specify a range of network prefixes. These keywords have the following values:

```
› prefix length < ge <= 32
```

```
› prefix length < le <= ge
```

- If you do not specify either **ge** or **le**, an exact match is expected. For example, if you issue the following command, only a route with a prefix length of 8 and a network address of 151.0.0.0 is permitted.

```
host1(config)#ip prefix-list abc permit 151.0.0.0/8
```

- Use the **no** version to remove the specified prefix list or the specified list entry.

match ip address

- Use with the **prefix-list** keyword to match routes that have a destination network number address that is permitted by the prefix list.

- Example

```
host1(config-route-map)#match ip address prefix-list abc
```

- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match ip next-hop

- Use with the **prefix-list** keyword to match routes that have a next-hop router address passed by the specified prefix list.
- Example

```
host1(config-route-map)#match ip next-hop prefix-list abc
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

Prefix Trees

A prefix tree is a nonsequential collection of permit and deny conditions that apply to IP addresses. Like a prefix list, the prefix tree specifies a base IP address and a length, the number of bits applied to the base to determine the network prefix. The tested address is matched against the prefix. The prefix tree also enables route summarization.

However, the prefix tree does not match addresses one by one in sequence against the listed conditions. The system performs a binary search against the tree structure of the entries. If the tested address is less than a particular entry, it branches one way to another test pair; if it is greater than the entry, it branches the other way to another mutually exclusive test pair. The system stops testing conditions when it finds the best match. If no conditions match, the system rejects the address. An empty prefix tree results in an automatic permit of the tested address.

The prefix tree provides a faster search methodology and matches the test address more closely than either the access list or the prefix list.

Use the **ip prefix-tree** command to define an IP prefix tree. Use the **prefix-tree** keyword with the **match ip address** or **match ip next-hop** commands to add a clause to a route map. Use the **match-set summary prefix-tree** command to specify the prefix tree that summarizes routes for a particular route map.

Using a Prefix Tree

The following example creates a prefix tree that permits routes with a prefix length of 24 or larger in the 10.10.2.0/24 network:

```
host1(config)#ip prefix-tree xyz permit 10.10.2.0/24
```

clear ip prefix-tree

- Use to clear all hit counts in the prefix trees, or the specified entry from the specified prefix tree. (The hit count is incremented +1 each time an entry matches.)
- Example

```
host1#clear ip prefix-tree xyz
```
- There is no **no** version.

ip prefix-tree

- Use to create a prefix tree for best route filtering; specifies a tree entry—a deny or permit clause for a network address.
- The prefix tree name can be up to 32 characters long.
- Example

```
host1(config)#ip prefix-tree boston42 permit 10.10.2.0/24
```
- Use the **no** version to remove the specified prefix tree or the specified tree entry.

match ip address

- Use with the **prefix-tree** keyword to match routes that have a destination network number address that is permitted by the prefix tree.
- Example

```
host1(config-route-map)#match ip address prefix-tree xyz
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match ip next-hop

- Use with the **prefix-tree** keyword to match routes that have a next-hop router address passed by the specified prefix tree.
- Example

```
host1(config-route-map)#match ip next-hop prefix-tree xyz
```
- Use the **no** version to delete the match clause from a route map or a specified value from the match clause.

match-set summary prefix-tree

- Use to specify the prefix tree that summarizes routes for a particular route map.
- Use the **ip prefix-tree** command to set the conditions of the prefix tree, including which routes to summarize and how many bits of the network address to preserve.
- Example

```
host1(config-route-map)#match-set summary prefix-tree dog3
```
- Use the **no** version to disable the use of the prefix tree by the route map.

Community Lists

A community is a logical group of prefixes that share some common attribute. Community members can be on different networks and in different autonomous systems. BGP allows you to define the community to which a prefix belongs. A prefix can belong to more than one community. The community attribute lists the communities to which a prefix belongs.

You can use communities to simplify routing policies by configuring the routing information that a BGP speaker will accept, prefer, or distribute to other neighbors according to community membership. When a route is learned, advertised, or redistributed, a BGP speaker can set, append, or modify the community of a route. When routes are aggregated, the resulting BGP update contains a community attribute that contains all communities from all of the aggregated routes (if the aggregate is an AS-set aggregate).

Several well-known communities have been predefined. Table 1-2 describes how a BGP speaker handles a route based on the setting of its community attribute.

Table 1-2 Action based on well-known community membership

If the well-known community is...	The BGP speaker...
no-export	Does not advertise the route beyond the BGP confederation boundary
no-advertise	Does not advertise the route to any peers, IBGP, or EBGP
local-as (also known as no-export-subconfed)	Does not advertise the route to any external peers
internet	Advertises this route to the Internet community; by default, all prefixes are members of the Internet community

In addition to the well-known communities, you can define local-use communities, also known as private communities or general communities. These communities serve as a convenient way to categorize groups of routes to facilitate the use of routing policies. The community attribute consists of four octets, but it is common practice to designate communities in the *AA:NN* format. The autonomous system number (*AA*) comprises the higher two octets, and the community number (*NN*) comprises the lower two octets. Both are expressed as decimal numbers. For example, if a prefix in AS 23 belongs to community 411, the attribute could be expressed as 23:411. Use the **ip bgp-community new-format** command to specify that the **show** commands display communities in this

format. You can also use a regular expression to specify the community attribute.

Use the **set community** command in route maps to configure the community attributes. You can add one or more communities to the attribute, or use the **list** keyword to add a list of communities to the attribute. By default, the community attribute is not sent to BGP peers. To send the community attribute to a neighbor, use the **neighbor send community** command.

A community list is a sequential collection of permit and deny conditions. Each condition describes the community number to be matched. If you issued the **ip bgp-community new-format** command, the community number is in *AA:NN* format; otherwise it is in decimal format (the hexadecimal octets converted to decimal).

The router tests the community attribute of a route against each condition in a community list. The first match determines whether the router accepts (the route is permitted) or rejects (the route is denied) a route having the specified community. Because the router stops testing conditions after the first match, the order of the conditions is critical. If no conditions match, the router rejects the route.

Consider the network structure shown in Figure 1-5.

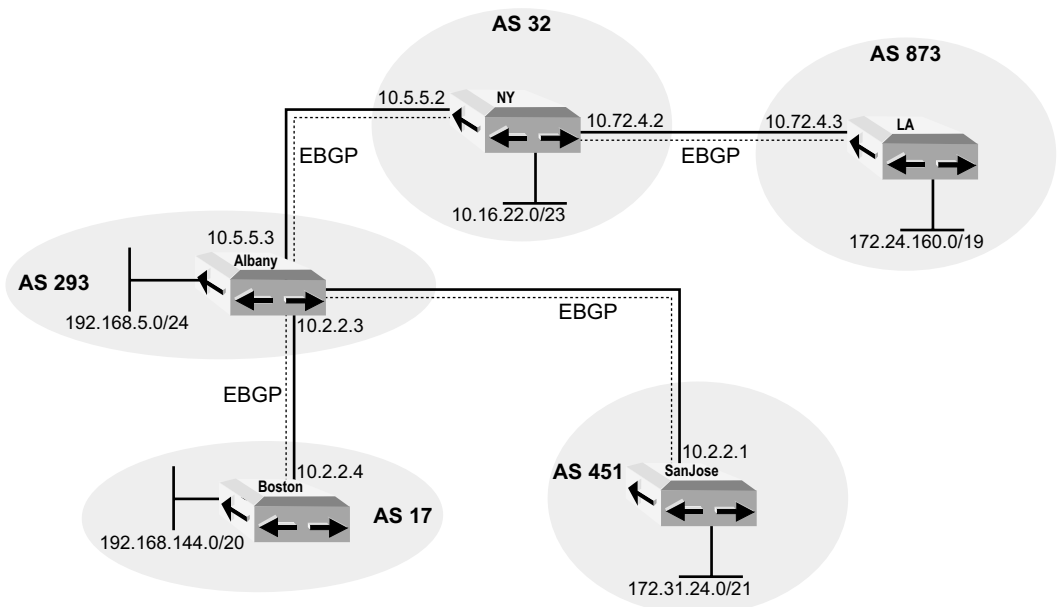


Figure 1-5 Community lists

Suppose you want router Albany to set metrics for routes that it forwards to router Boston based on the communities to which the routes belong. You can create community lists and filter the routes with a route map that matches on the community list. The following commands demonstrate how to configure router Albany:

```
host1(config)#router bgp 293
host1(config-router)#neighbor 10.5.5.2 remote-as 32
host1(config-router)#neighbor 10.2.2.1 remote-as 451
host1(config-router)#neighbor 10.2.2.4 remote-as 17
host1(config-router)#neighbor 10.2.2.4 route-map commtrc out
host1(config-router)#exit
host1(config)#route-map commtrc permit 1
host1(config-route-map)#match community 1
host1(config-route-map)#set metric 20
host1(config-route-map)#exit
host1(config)#route-map commtrc permit 2
host1(config-route-map)#match community 2
host1(config-route-map)#set metric 75
host1(config-route-map)#exit
host1(config)#route-map commtrc permit 3
host1(config-route-map)#match community 3
host1(config-route-map)#set metric 85
host1(config-route-map)#exit
host1(config)#ip community-list 1 permit 25
host1(config)#ip community-list 2 permit 62
host1(config)#ip community-list 3 permit internet
```

Community list 1 comprises routes with a community of 25; their metric is set to 20. Community list 2 comprises routes with a community of 62; their metric is set to 75. Community 3 catches all remaining routes by matching the internet community; their metric is set to 85.

ip bgp-community new-format

- Use to specify that communities must be displayed in *AA:NN* format, where *AA* is a number that identifies the autonomous system and *NN* is a number that identifies the community within the autonomous system.
- Example

```
host1(config)#ip bgp-community new-format
```
- Use the **no** version to restore the default display.

ip community-list

- Use to create a community list for BGP and control access to it. The list name can be up to 32 characters long.
- A route can belong to any number of communities, so a community list can have many entries comprising many communities.
- You can specify one or more community values when you create a community list. A clause in a route map that includes a list having more than one value matches only a route having all of the values; that is, the multiple values are logical ANDed.
- You can specify community values with a number or a regular expression.
- Example

```
host1(config)#ip community-list 1 permit 100:2 100:3 100:4
host1(config)#route-map marengo permit 10
host1(config-route-map)#match community 1
```

A route matches this community list only if it belongs to at least all three communities in community list 1: communities 100:2, 100:3, and 100:4.

- Use the **no** version to remove the specified community list, including all list entries.

neighbor send-community

- Use to specify that a community attribute should be sent to a BGP neighbor.
- If you specify a BGP peer group by using the *peer-group-name* argument, all the members of the peer group will inherit the characteristic configured with this command.
- Use the **no** version to specify that common attributes should not be sent to a BGP neighbor.

set community

- Use to set the community attribute in BGP updates.
- You can specify a community list number in the range 1–4294967295, or in the new community format of *AA:NN*, or one of the following well-known communities:
 - › **local-as** – prevents advertisement outside of the local AS
 - › **no-advertise** – prevents advertisement to any peer
 - › **no-export** – prevents advertisement beyond the BGP confederation boundary
- Alternatively, you can use the **list** keyword to specify the name of a community list that you previously created with the **ip community-list** command.
- Supported for inbound, outbound, and redistribution route maps.
- Example

```
host1(config)#route-map 1
host1(config-route-map)#set community no-advertise
```

- Use the **none** keyword to remove the community attribute from a route.
- Use the **no** version to delete the entry.

Extended Community Lists

The system supports the BGP extended community attribute defined in the Internet draft, BGP Extended Communities Attribute – draft-ietf-idr-bgp-ext-communities-05.txt (November 2002 expiration). This attribute enables the definition of a new type of IP extended community and extended community list unrelated to the community list that uses regular expressions.



Note: IETF drafts are valid for only 6 months from the date of issuance. They must be considered as works in progress. Please refer to the IETF Web site at <http://www.ietf.org> for the latest drafts.

BGP speakers can use the extended community attribute to control routes much like they use the community attribute to determine routes they accept, reject, or redistribute. A BGP speaker can append the extended community attribute to a route that does not have the attribute before advertising the route. For routes that do have the attribute, BGP can modify the attribute.

ip extcommunity-list

- Use to create an extended community list for BGP and control access to it.
- A route can belong to any number of communities, so an extended community list can have many entries comprising many communities.
- You can specify one or more community values when you create an extended community list. A clause in a route map that includes a list having more than one value matches only a route having all of the values; that is, the multiple values are logical ANDed.
- Use the **rt** keyword to specify a route target community, which consists of one or more routers that can receive a set of routes advertised by BGP that carry the extended community attribute.
- Use the **soo** keyword to specify a site-of-origin community, which consists of one or more routers that inject into BGP a set of routes that carry the extended community attribute.
- Example

```
host1(config)#ip extcommunity-list boston1 permit rt 100:2
                rt 100:3 rt 100:4
host1(config)#route-map marengo permit 10
host1(config-route-map)#match extcommunity boston1
```

A route matches this community list only if it belongs to at least all three communities in extended community list boston1: communities 100:2, 100:3, and 100:4.

- Use the **no** version to remove a single extended community list entry if **permit** or **deny** and a *path-expression* are specified. Otherwise, the entire community list is removed.

match extcommunity

- Use to match an extended community list in a route map.
- You can specify one or more extended community list names in a match clause. If you specify more than one extended community list, the lists are logically ORed.
- Example

```
host1(config-route-map)#match extcommunity topeka10
```
- Use the **no** version to remove the match clause from a route map or a specified value from the match clause.

set extcommunity

- Use to set the extended community attributes in a route map for BGP updates.
- Use the **rt** keyword to specify a route target community, which consists of one or more routers that can receive a set of routes advertised by BGP that carry the extended community attribute.
- Use the **soo** keyword to specify a site-of-origin community, which consists of one or more routers that inject into BGP a set of routes that carry the extended community attribute.
- You can specify both a route target community and a site-of-origin community at the same time in a set clause without overwriting each other.
- Example

```
host1(config)#route-map 1  
host1(config-route-map)#set extcommunity rt 10.10.10.2:325
```
- Use the **no** version to remove the set clause from the route map.

show ip extcommunity-list

- Use to display information about a specific extended community list or all extended community lists.
- Example

```
host1#show ip extcommunity-list  
IP Extended Community List dresden1:  
    permit soo 10.10.10.10:15  
IP Extended Community List bonn:  
    deny rt 12:12
```

Using Regular Expressions

You can use regular expressions when defining AS-path access lists and community lists to more easily filter routes. A regular expression uses special characters—often referred to as metacharacters—to define a pattern that is compared with an input string.

AS-path Lists

For an AS-path access list, the input string is the AS path of the routes to which the list is applied via the **route-map** or **neighbor filter-list** commands. If the AS path matches the regular expression in the access list, then the route matches the access list.

Example The following commands apply access list 1 to routes inbound from BGP peer 10.5.5.2. Access list 1 uses a regular expression to deny routes originating in AS 32.

```
host1(config-router)#neighbor 10.5.5.2 remote-as 32
host1(config-router)#neighbor 10.5.5.2 filter-list 1 in
host1(config-router)#exit
host1(config)#ip as-path access-list 1 deny 32$
```

Community Lists

For a community list, the input string is the community attribute of the routes to which the list is applied via a **route-map** command. If the community attribute matches the regular expression in the community list, then the route matches the community list.

Example The following commands apply route map 5 to routes forwarded to BGP peer 10.5.5.4. Route map 5 uses a regular expression to match community numbers ending with 305, setting the weight of matching routes to 150.

```
host1(config-router)#neighbor 10.5.5.4 remote-as 425
host1(config-router)#neighbor 10.5.5.4 route-map 5 out
host1(config-router)#exit
host1(config)#route-map 5 permit 10
host1(config-route-map)#match community 305$
host1(config-route-map)#set weight 150
```

Community Numbers

When you use a regular expression to match a community number, use the appropriate format for the community number in the community list. If you issued the **ip bgp-community new-format** command, the community number has the format *AA:NN* where *AA* is a number that identifies the autonomous system, and *NN* is a number that identifies the community within the autonomous system. However, if you did not issue this command, the community number is an integer in the range 1–4294967295.

Metacharacters

Each regular expression consists of one or more metacharacters and zero or more complete or partial AS or community numbers. Table 1-3 describes the metacharacters supported for regular expression pattern-matching.

Table 1-3 Supported regular expression metacharacters

Metacharacter	Description
^	Matches the beginning of the input string. Alternatively, when used as the first character within brackets—[[^]]—matches any number except the ones specified within the brackets.
\$	Matches the end of the input string.
.	Matches any single character, including white space.
*	Matches 0 or more sequences of the immediately previous character or pattern.
+	Matches 1 or more sequences of the immediately previous character or pattern.
?	Matches 0 or 1 sequence of the immediately previous character or pattern.
()	Specifies patterns for multiple use when followed by one of the multiplier metacharacters: asterisk *, plus sign +, or question mark ?
[]	Matches any enclosed character; specifies a range of single characters.
– (hyphen)	Used within brackets to specify a range of AS or community numbers.
_ (underscore)	Matches a ^, a \$, a comma, a space, a {, or a }. Placed on either side of a string to specify a literal and disallow substring matching. Numerals enclosed by underscores can be preceded or followed by any of the characters listed above.
	Matches characters on either side of the metacharacter; logical OR.

Using Metacharacters as Literal Tokens

You can remove the special meaning of a metacharacter by preceding it with a backslash (\). Such a construction denotes that the metacharacter is *not* treated as a metacharacter for that regular expression. It is simply a character or token with no special meaning, just as a numeral has no special meaning. The backslash applies only to the character immediately following it in the regular expression.

On the ERX system, you are likely to do this only for the parentheses characters, (or). BGP indicates a segment of an AS path that is of type

AS-confed-set or AS-confed-seq by enclosing that segment within parentheses.

Example The following AS-path access list uses a regular expression to match routes having an AS-path attribute that *begins* with any AS-confed-set or AS-confed-seq:

```
host1(config)#ip as-path access-list 1 permit ^\((
```

The following AS-path access list uses a regular expression to match routes having an AS-path attribute that *ends* with any AS-confed-set or AS-confed-seq:

```
host1(config)#ip as-path access-list 1 permit \)$
```

The following AS-path access list uses a regular expression to match routes having an AS-path attribute that *includes* the specific AS-confed-set or AS-confed-seq, (100 200):

```
host1(config)#ip as-path access-list 1 permit \((100 200\)
```

Regular Expression Examples

Table 1-4 shows some representative regular expressions that might be used in an AS-path access list or community list along with sample attribute values that match or do not match the regular expression.

Table 1-4 Sample regular expressions

Regular Expression	Matches Any AS-path or Community Attribute That...
^12	Begins with 12, such as the following: 12 23 42 12 629 12 1245 19 but not 58 12 7
[^12]	Includes any numeral except 1 or 2, such as the following: 44 73 46 5 69 8 but not 1145 19 12 2 49
305\$	Ends with 305, such as the following: 89 611 305 305 42 305 19 1305 6666:305

Troubleshooting Routing Policy

You can turn on debugging for routing policy by issuing the **log severity debug ipRoutePolicy** command from Global Configuration mode. You can specify different levels of severity for ipRoutePolicy. See *ERX System Basics Configuration Guide, Chapter 4, Managing the System*, for more information on using **log** commands for troubleshooting.

Monitoring Routing Policy

You can monitor the following aspects of routing policy using **show** commands:

- Access lists

```
host1#show access-list
host1#show ip as-path access-list
```

- Community lists

```
host1#show ip community-list
```

- Prefix lists

```
host1#show ip prefix-list
```

- Prefix trees

```
host1#show ip prefix-tree
```

- Protocols

```
host1#show ip protocols
```

- Redistribution policies

```
host1#show ip redistribute
```

- Routes

```
host1#show ip route
```

- Interfaces and next hops

```
host1#show ip route slot 5 192.168.5.4
```

- Route maps

```
host1#show route-map
```

- Static routes

```
host1#show ip static
```

- IP traffic

```
host1#show ip traffic
```

You can use the output filtering feature of the **show** command to include or exclude lines of output based on a text string you specify. Refer to *ERX System Basics Configuration Guide, Chapter 2, Command Line Interface*, for details.

show access-list

- Use to display information about access lists.
- The displayed information includes the instances of each access list.
- Use the **detail** keyword to display the automatically assigned element ID for each access-list entry. Only rules that you explicitly create have element IDs.
- Example

```
host1#show access-list
IP Access List 1:
  permit ip host 172.31.192.217 any
  permit ip 12.40.0.0 0.0.0.3 any
  deny ip any any
IP Access List 2:
  permit ip 172.19.0.0 0.0.255.255 any
  deny ip 0.0.0.0 255.255.255.255 any
IP Access List 10:
  permit ip any any
IP Access List 11:
  deny ip any any
host1#show access-list detail
IP Access List 1:
  1: permit ip host 172.31.192.217 any
  2: permit ip 12.40.0.0 0.0.0.3 any
  deny ip any any
```

show ip as-path-access-list

- Use to display information about AS-path access lists.
- Example

```
host1#show ip as-path-access-list
AS Path Access List 1:
  permit .*
AS Path Access List 2:
  deny .*
AS Path Access List 3:
  permit _109_
  deny .*
AS Path Access List 4:
  permit _109$
```

```

deny .*
AS Path Access List 10:
deny _109$
permit ^108_
deny .*

```

show ip community-list

- Use to display community list information.
- Display varies based on whether you issued the **ip bgp community new-format** command.
- Example 1. If you did not issue the **ip bgp community new-format** command, the display appears as follows:

```

host1#show ip community-list
Community List 1:
  permit 81200109
  permit 81200110
  permit 81200108
Community List 2:
  deny 81200109
  permit 81200110
  permit 81200108
Community List 4:
  permit local-as
Community List 5:
  permit no-advertise
Community List 6:
  permit no-export
Community List 7:
  permit internet

```

- Example 2. If you did issue the **ip bgp community new-format** command, the display appears as follows:

```

host1#show ip community-list
Community List 1:
  permit 1239:1005
  permit 1239:1006
  permit 1239:1004
Community List 2:
  deny 1239:1005
  permit 1239:1006
  permit 1239:1004
Community List 4:
  permit local-as
Community List 5:
  permit no-advertise
Community List 6:

```

```
    permit no-export
Community List 7:
    permit internet
```

show ip prefix-list

- Use to display information about the prefix lists currently configured on the system.
- Example

```
host1#show ip prefix-list
Prefix-list with the last deletion/insertion: def
ip prefix-list name abc: 4 entries
  seq 5 permit 192.168.0.0/16 le 24
  seq 10 permit 192.178.0.0/16 le 24
  seq 15 deny 195.178.0.0/16 le 24
  seq 20 deny 195.178.0.0/16 le 32
ip prefix-list name def: 1 entries
  seq 5 deny 192.170.0.0/16
```

show ip prefix-list summary | detail

- Use to display summarized or detailed information about the prefix lists currently configured on the system.
- Example

```
host1#show ip prefix-list summary
Total memory used for prefix-list: 310 bytes
Prefix-list with the last deletion/insertion: def
ip prefix-list name abc:
  count: 4, range entries: 4, sequences: 5-20
ip prefix-list name def:
  count: 1, range entries: 0, sequences: 5-5
```

show ip prefix-tree

- Use to display information about the prefix trees currently configured on the system.
- Example

```
host1#show ip prefix-tree
Prefix-tree with the last deletion/insertion: t_abc5
ip prefix-tree name t_abc1: 1 entries
  permit 108.243.0.0/16
ip prefix-tree name t_abc2: 3 entries
  permit 101.10.254.0/24
  permit 102.10.248.0/21
  permit 103.10.192.0/18
  permit 108.109.0.0/16
  permit 108.109.241.0/24
```

```
ip prefix-tree name t_abc3: 1 entries
deny 108.0.0.0/8
```

show ip prefix-tree summary | detail

- Use to display summarized or detailed information about the prefix trees currently configured on the system.
- Example

```
host1#show ip prefix-tree summary
Total memory used for prefix-tree: 860 bytes
Prefix-tree with the last deletion/insertion: t_abc5
ip prefix-tree name t_abc1:
count: 1
ip prefix-tree name t_abc2:
count: 5
ip prefix-tree name t_abc3:
count: 1
```

show ip protocols

- Use to display detailed information about the protocols currently configured on the system.
- Use the **summary** keyword to display only a list of the configured protocols.
- For field descriptions, see the **show** commands for the individual routing protocols in their respective *Configuration Guide* chapters.
- Example

```
host1#show ip protocols
Routing Protocol is "bgp 1"

Default local preference is 100
IGP synchronization is enabled
Always compare MED is disabled
Router flap damping is disabled
Administrative Distance: external 20 internal 200 local
200
Neighbor(s):
No neighbors are configured
Routing for Networks:

Routing Protocol is "ospf 255" with Router ID 100.100.100.1
Distance is 110

Address Summarization:
None
Routing for Networks:

Routing Protocol is "rip"
Router Administrative State: enable
```

```
System version RIP1: send = 1, receive = 1 or 2
Update interval: 30 seconds
Invalid after: 180 seconds
hold down time: 120 seconds
flushed interval: 300 seconds
Filter applied to outgoing route update is not set
Filter applied to incoming route update is not set
No global route map
Distance is 120
Interface          Tx    Rx    Auth

Routing for Networks:
  10.2.1.0/255.255.255.0
```

show ip redistribute

- Use to display configured route redistribution policy.
- Field descriptions
 - › To – protocol into which routes are distributed
 - › From – protocol from which routes are distributed
 - › status – redistribution status
 - › route map number – number of the route map

- Example

```
host1#show ip redistribute
To ospf, From static is enabled with route map 4
To ospf, From connected is enabled with route map 3
```

show ip route

- Use to display the current state of the routing table, including routes not used for forwarding.
- You can display all routes, a specific route, all routes beginning with a specified address, routes for a particular protocol (BGP, IS-IS, OSPF, or RIP), locally connected routes, internal control routes, static routes, or summary counters for the routing table.
- Field descriptions
 - › Prefix – IP address prefix
 - › Length – prefix length
 - › Type – protocol type
 - › Next Hop – IP address of the next hop
 - › Dist – distance metric for the route
 - › Met – number of hops
 - › Intf – interface type and interface specifier

- Example 1

```
host1#show ip route
```

```
Protocol/Route type codes:
```

```
I1- ISIS level 1, I2- ISIS level2,  
I- route type intra, IA- route type inter, E- route type external,  
i- metric type internal, e- metric type external,  
O- OSPF, E1- external type 1, E2- external type2,  
N1- NSSA external type1, N2- NSSA external type2
```

Prefix/Length	Type	Next Hop	Dist/Met	Intf
172.16.2.0/24	Bgp	192.168.1.102	20/1	fastEthernet0/0
10.10.0.112/32	Static	192.168.1.1	1/1	fastEthernet0/0
10.1.1.0/24	Connect	10.1.1.1	0/1	atm3/0.100

- Example 2

```
host1#show ip route static
```

```
Protocol/Route type codes:
```

```
I1- ISIS level 1, I2- ISIS level2,  
I- route type intra, IA- route type inter, E- route type external,  
i- metric type internal, e- metric type external,  
O- OSPF, E1- external type 1, E2- external type2,  
N1- NSSA external type1, N2- NSSA external type2
```

Prefix/Length	Type	Next Hop	Dist/Met	Intf
10.10.0.112/32	Static	192.168.1.1	1/1	fastEthernet0/0

- Example 3

```
host1#show ip route summary
```

```
5 total routes, 720 bytes in route entries  
0 isis routes  
0 rip routes  
1 static routes  
1 connected routes  
0 bgp routes  
0 ospf routes  
3 other internal routes  
0 access routes  
0 internally created access host routes
```

```
Last route added/deleted: 0.0.0.0/0 by StaticLow  
At THU MAR 09 2000 05:22:49 UTC
```

show ip route slot

- Use to display the interface and next hop for an IP address in the routing table of a line module specified by the slot it occupies.
 - › *slotNumber* – number of slot containing the line module for which the information is displayed
 - › *ipAddress* – IP address to look up in the routing table
- Field descriptions
 - › IP address – address reachable via the interface
 - › Interface – interface type and specifier associated with the IP address; displays “Local Interface” if a special interface index is present in the routing table for special IP addresses, such as broadcast addresses
 - › Next Hop – next hop to reach the IP address; displays “---” if no next hop is associated with the IP address

• Example 1

```
host1#show ip route slot 6 10.10.0.231
  IP address      Interface          Next Hop
  -----
  10.10.0.231    fastEthernet 6/0  10.10.0.231
```

• Example 2

```
host1#show ip route slot 9 90.248.1.2
  IP address      Interface          Next Hop
  -----
  90.248.1.2     serial9/23:2      ---
```

• Example 3

```
host1#show ip route slot 9 90.249.255.255
  IP address      Interface          Next Hop
  -----
  90.249.255.255 Local Interface    ---
```

show ip static

- Use to display the status of static routes in the routing table.
- You can specify an optional IP mask that filters specific routes.
- Field descriptions
 - › Prefix – IP address prefix
 - › Length – prefix length
 - › Next Hop – IP address of the next hop
 - › Met – number of hops
 - › Dist – administrative distance or weight assigned to the route
 - › Tag – tag value assigned to the route
 - › Intf – interface type and interface specifier
- Example

```
host1#show ip static
Prefix/Length  Next Hop:      Met: Dist:  Tag:  Intf:
```

10.2.0.0/24	192.168.1.1	1	1	0	ethernet6/0
10.2.1.0/24	192.168.1.1	1	1	1	ethernet6/0
172.31.1.48/32	172.18.2.2	1	1	0	atm5/1.1

show ip traffic

- Use to display statistics about IP traffic.
- Field descriptions
 - › IP Statistics Rcvd:
 - Router Id – router ID number
 - total – number of frames received
 - local destination – frames with this router as their destinations
 - hdr errors – number of packets containing header errors
 - addr errors – number of packets containing addressing errors
 - unkn proto – number of packets received containing unknown protocols
 - discards – number of discarded packets
 - › IP Statistics Frags:
 - reassembled – number of reassembled packets
 - reasm timed out – number of reassembled packets that timed out
 - reasm req – number of requests for reassembly
 - reasm fails – number of reassembly failures
 - frag ok – number of fragmented packets reassembled successfully
 - frag fail – number of fragmented packets reassembled unsuccessfully
 - frag creates – number of packets created by fragmentation
 - › IP Statistics Sent:
 - forwarded – number of packets forwarded
 - generated – number of packets generated
 - out disc – number of outbound packets discarded
 - no routes – number of packets that could not be routed
 - routing discards – number of packets that could not be routed that were discarded
 - › IP Statistics Route:
 - routes in table – number of routes in the routing table
 - timestamp req – number of requests for a timestamp
 - timestamp rpy – number of replies to timestamp requests
 - addr mask req – number of address mask requests
 - addr mask rpy – number of address mask replies
 - › ICMP Statistics Rcvd:
 - total – total number of ICMP packets received
 - errors – number of error packets received
 - dst unreach – number of packets received with destination unreachable
 - time exceed – number of packets received with time-to-live exceeded
 - param probs – number of packets received with parameter errors

- src quench – number of source quench packets received
 - redirects – number of receive packet redirects
 - echo req – number of echo request (PING) packets
 - echo rpy – number of echo replies received
 - timestamp req – number of requests for a timestamp
 - timestamp rpy – number of replies of timestamp requests
 - addr mask req – number of mask requests sent
 - addr mask rpy – number of mask replies sent
- › ICMP Statistics Sent:
- total – total number of ICMP packets sent
 - errors – number of error packets sent
 - dest unreach – number of packets sent with destination unreachable
 - time excd – number of packets sent with time-to-live exceeded
 - param prob – number of packets sent with parameter errors
 - src quench – number of source quench packets sent
 - redirects – number of send packet redirects
 - echo req – number of echo request (PING) packets
 - echo rpy – number of echo replies received
 - timestamp req – number of requests for a timestamp
 - timestamp rpy – number of replies to timestamp requests
 - addr mask req – number of address mask requests
 - addr mask rpy – number of address mask replies
- › UDP Statistics Rcvd:
- total – total number of UDP packets received
 - checksum – number of checksum error packets received
 - no port – number of packets received for which no application listener was listening on the destination port
- › UDP Statistics Sent:
- total – total number of UDP packets sent
 - errors – number of error packets sent
- › TCP Global Statistics Connections:
- attempted – number of outgoing TCP connections attempted
 - accepted – number of incoming TCP connections accepted
 - established – number of TCP connections established
 - dropped – number of TCP connections dropped
 - closed – number of TCP connections closed
- › TCP Global Statistics Rcvd:
- total pkts – total number of TCP packets received
 - in-sequence pkts – number of packets received in sequence
 - bytes – number of bytes received
 - chksum err pkts – number of checksum error packets received

- authentication err pkts – number of authentication error packets received
- bad offset pkts – number of packets received with bad offsets
- short pkts – number of short packets received
- duplicate pkts – number of duplicate packets received
- out of order pkts – number of packets received out of order
- › TCP Global Statistics Sent:
 - total pkts – total number of TCP packets sent
 - data pkts – number of data packets sent
 - bytes – number of bytes sent
 - retransmitted pkts – number of packets retransmitted
 - retransmitted bytes – number of retransmitted bytes
- › OSPF Statistics – not supported for this version of the system
- › IGMP Statistics – not supported for this version of the system
- › ARP Statistics – not supported for this version of the system
- Example

```
host1#show ip traffic
IP statistics: Router Id: 172.31.192.217
  Rcvd: 97833 total, 171059 local destination
        0 hdr errors, 0 addr errors
          167 unkn proto, 0 discards
        Frags: 4 reassembled, 30 reasm timed out, 8 reasm req
          0 reasm fails, 145 frag ok, 0 frag fail
          290 frag creates
  Sent: 15 forwarded, 25144 generated, 0 out disc
        0 no routes,0 routing discards
  Route: 57680 routes in table
        0 timestamp req, 0 timestamp rpy
        0 addr mask req, 0 addr mask rpy

ICMP statistics:
  Rcvd: 561 total, 0 errors, 15 dst unreach
        0 time exceed, 0 param probs, 0 src quench
        0 redirects, 0 echo req, 0 echo rpy
        0 timestamp req, 0 timestamp rpy
        0 addr mask req, 0 addr mask rpy
  Sent: 0 total, 0 errors, 0 dest unreach
        0 time excd, 0 param prob, 0 src quench
        0 redirects, 0 echo req, 0 echo rpy

UDP Statistics:
  Rcvd: 93326 total, 0 checksum errors, 90610 no port
  Sent: 0 total, 0 errors
```

```
TCP Global Statistics:
  Connections: 7358 attempted, 4 accepted, 7362 established
               0 dropped, 14718 closed
  Rcvd: 75889 total pkts, 53591 in-sequence pkts, 3120283
        bytes
        0 chksum err pkts, 0 authentication err pkts, 0 bad
        offset
        0 short pkts, 0 duplicate pkts, 0 out of order pkts
  Sent: 82318 total pkts, 44381 data pkts, 656321 bytes
        34 retransmitted pkts, 487 retransmitted bytes

OSPF Statistics:

IGMP Statistics:

ARP Statistics:
```

show route-map

- Use to display the configured route maps.
- The displayed information includes the instances of each access list such as **match** and **set** commands.
- Example

```
host1(config)#route-map 1 permit 10
host1(config-route-map)#match community 44
host1(config-route-map)#set local-pref 400
host1(config-route-map)#exit
host1(config)#exit
host1#show route-map 1
route-map 1, permit, sequence 10
  Match clauses:
    match community 44
  Set clauses:
    set local-pref 400
```

