

Chapter 7

Configure AS Path Regular Expressions and Communities

For routing policies that apply to BGP, you can do the following:

Define AS Path Regular Expressions to Use in Routing Policies on page 65

Define Communities to Use in Routing Policies on page 69

Define AS Path Regular Expressions to Use in Routing Policies

A BGP AS path is the route to a destination. It consists of the AS numbers of all the routers a packet must go through to reach that destination. You can define routing policy matches based on the AS path. To do this, you create a named AS path regular expression, then include it in a policy with the as-path match condition (described in Table 2 on page 38).

To create a named AS path regular expression, include the as-path statement at the [edit policy-options] hierarchy level:

```
[edit policy-options]
as-path name regular-expression;
```

To include the AS path regular expression in a policy, include the as-path match condition in the from or to statement:

```
[edit policy-options]
as-path name regular-expression;
policy-statement policy-name {
  term term-name {
    from {
      (match-conditions | (route-filter destination-prefix match-type <actions>));
    }
    to {
      match-conditions;
    }
  }
}
```

The name identifies the regular expression. It can contain letters, numbers, and hyphens (-), and can be up to 255 characters. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

To include the AS path regular expression in a policy, include the as-path match condition in the from or to statement:

```
[edit policy-options]
as-path name regular-expression;
policy-statement policy-name {
  term term-name {
    from {
      (match-conditions | (route-filter destination-prefix match-type <actions>));
    }
    to {
      match-conditions;
    }
  }
}
```

regular-expression defines the regular expression used to match the AS path. It consists of two components, which you specify in the following format:

term <operator>

term—Identifies the AS. You can specify it in one of the following ways:

AS number.

Wildcard character—Matches any single AS number. The wildcard character is a period (.).

AS path—To group the path into a single element, enclose it in parentheses. (Do not include parentheses with nothing inside of them.) Grouping the regular expression in this way allows you to perform a common operation on the group as a whole. The grouped path can itself include operators.

operator—(Optional) Operator that defines pattern matching operations to apply to the term. You define the operations using regular expressions. Table 7 lists some of the regular expression operators you can use. You place operators immediately after *term* with no intervening space, except for the pipe (|) operator, which you place between two terms.

AS path regular expressions implement the extended (modern) regular expressions as defined in POSIX 1003.2 They are identical to the UNIX regular expressions with the following exceptions:

Each AS number is analyzed as a single regular expression, not as a sequence of individual characters.

A regular expression matches a route only if the AS path in the route exactly matches *regular-expression*. The equivalent UNIX regular expression is *^regular-expression\$*. For example, the AS path regular expression 1234 is equivalent to the UNIX regular expression *^1234\$*.

You can specify one or more term-operator pairs in a single regular expression.

Table 8 shows examples of how to define regular expressions to match AS paths.

Table 7: Regular Expression Operators

Operator	Match...
{ <i>m,n</i> }	At least <i>m</i> and at most <i>n</i> repetitions of <i>term</i> . Both <i>m</i> and <i>n</i> must be positive integers, and <i>m</i> must be smaller than <i>n</i> .
{ <i>m</i> }	Exactly <i>m</i> repetitions of <i>term</i> . <i>m</i> must be a positive integer.
{ <i>m</i> ,}	<i>m</i> or more repetitions of <i>term</i> . <i>m</i> must be a positive integer.
*	Zero or more repetitions of <i>term</i> . This is equivalent to {0,}.
+	One or more repetitions of <i>term</i> . This is equivalent to {1,}.
?	Zero or one repetition of <i>term</i> . This is equivalent to {0,1}.
	One of the two terms on either side of the pipe.
^	Character at the beginning of a BGP community name. You cannot use this operator in AS path regular expressions.
\$	Character at the end of a BGP community name. You cannot use this operator in AS path regular expressions.
[]	Range of letters or digits. To separate the start and end of a range, use a hyphen (-). You cannot use this operator in AS path regular expressions.
()	A group of terms.

Table 8: Examples of Defining AS Path Regular Expressions

AS Path to Match	Regular Expression	Example Matches
Exactly one occurrence of AS number 1234 in the path	1234	1234
Zero or more occurrences of AS number 1234	1234*	1234 1234 1234 1234 1234 1234 nothing
Zero or one occurrence of AS number 1234	1234? or 1234{0,1}	1234 nothing
One through four occurrences of AS number 1234	1234{1,4} or 1234{1,4}	1234 1234 1234 1234 1234 1234 1234 1234 1234 1234
One through four occurrences of AS number 12 followed by one occurrence of AS number 34	12{1,4}34	12 34 12 12 34 12 12 12 12 34
Range of AS numbers to match a single AS number	123-125	123 or 124 or 125

AS Path to Match	Regular Expression	Example Matches
Path whose second AS number must be 56 or 78	. 56 . 78 or . (56 78)	1234 56 34 78
Path whose second AS number might be 56 or 78	. (56 78)?	1234 1234 56 34 78
Path of any length whose second AS number can be anything, including nonexistent	1234 .* or 1234 .{0,}	1234 1234 5678 1234 5 6 7 8
One occurrence of the AS path 1, 2, 3	1 2 3	1 2 3
One occurrence of the AS numbers 1 and 2, followed by one or more occurrences of the number 3	1 2 3+ ;	1 2 3 1 2 3 3 1 2 3 3 3
One or more occurrences of AS number 1, followed by one or more occurrences of AS number 2, followed by one or more occurrences of AS number 3	1+ 2+ 3+	1 2 3 1 1 2 3 1 1 2 2 3 1 1 2 2 3 3
Path of any length that begins with AS numbers 4, 5, 6	4 5 6 .*	4 5 6 4 5 6 7 8 9
Path of any length that ends with AS numbers 4, 5, 6	.* 4 5 6	4 5 6 1 2 3 4 5 6

Examples: Create AS Path Regular Expressions in Routing Policies

Exactly match routes with the AS path 1234 56 78 9 and accept them:

```
[edit]
policy-options {
  as-path wellington "1234 56 78 9";
  policy-statement from-wellington {
    term term1 {
      from as-path wellington;
      then preference 200 accept;
    }
    term term2 {
      then reject;
    }
  }
}
```

Match alternate paths to an AS:

```
[edit]
policy-options {
  as-path wellington-alternate "1234{1,6} (56|47)? (78|101|112)* 9+";
  policy-statement from-wellington {
    from as-path wellington-alternate;
    then preference 200;
  }
}
```

Match a route with an AS-path of 123, 124, or 125:

```
[edit]
policy-options {
  as-path wellington "123-125";
}
```

Define Communities to Use in Routing Policies

The basic unit of matching in a community regular expression is a character.

Communities are an unordered property on a route. For example, 1:2 3:4 is the same as 3:4 1:2.

Community policy works as follows:

For each route that passes through a policy, the route is evaluated against each named community in a policy from clause. If one of these evaluations is successful, the policy then action is taken.

The evaluation of a named community involves evaluating the route against each member of the named community. All these evaluations must be successful for the named community evaluation to be successful.

The evaluation against a community member involves performing a character-based regular expression match of each community on the route against the community member. Only one community from the route must match for the community member evaluation to be successful.

Each member in a named community is either a literal community value or a regular expression that must match a community value for that route to be considered part of that community. For example:

```
[edit]
policy-options {
  policy-statement one {
    from {
      community [ comm-one comm-two ];
    }
  }
  community comm-one [ 1:2 "^4:(5|6)$" ];
  community comm-two [ 7:8 9:10 ];
}
```

To match policy one, the route must match either comm-one or comm-two.

To match comm-one, the route must have a community that matches 1:2 and a community that matches 4:(5|6).

To match comm-two, the route must have a community that matches 7:8 and a community that matches 9:10.

To match routes that are tagged with a particular set of communities and to apply the following logic:

```
include communities (1:1 AND 2:2 AND (3:3 OR 4:4 OR 5:5))
```

configure the community as follows:

```
community comm-1 members [ 1:1 2:2 "(3:3|4:4|5:5" );
```

You can configure community and extended community attributes to be included in BGP update messages.

A BGP community is a group of destinations that share a common property. You can define routing policy matches based on the BGP communities. To do this, you create a named community and then include it in a policy with the community match condition (described in Table 2 on page 36).

The BGP extended communities attribute provides a larger range (8 octets) for grouping or categorizing communities for applying routing decisions, such as acceptance, rejection, preference, or redistribution. The community attribute is only 4 octets.

The community-id format varies according to the type of attribute that you use. The BGP community attribute format is *as-number :community-value*. The BGP extended communities attribute format is *type:administrator:assigned-number*.

To define communities, you can do the following:

- Configure the BGP Community Attribute on page 68

- Configure the BGP Extended Communities Attribute on page 73

Configure the BGP Community Attribute

To create a named community and define the community members, include the community statement at the [edit policy-options] hierarchy level:

```
[edit policy-options]
  community name members [community-ids];
```

The name identifies the community. It can contain letters, numbers, and hyphens (-), and can be up to 255 characters long. To include spaces in the name, enclose the entire name in quotation marks (double quotes).

community-ids defines one or more members of the community. It consists of two components, which you specify in the following format:

as-number :community-value

as-number—AS number of the community member. It can be a value from 0 through 65535.

community-value—Identifier of the community member. It can be a number from 0 through 65535.

You also can specify *community-id* as one of the following well-known community names, which are defined in RFC 1997:

no-export—Routes in this community must not be advertised outside a BGP confederation boundary.

no-advertise—Routes in this community name must not be advertised to other BGP peers.

no-export-subconfed—Routes in this community must not be advertised to external BGP peers, including peers in other members' ASs inside a BGP confederation.

When specifying *community-id*, you can use UNIX-style regular expressions to specify the AS number and the member identifier. A regular expression consists of two components, which you specify in the following format:

term<*operator*>

term—Identifies the string to match.

operator—(Optional) Operator that defines pattern matching operations to apply to the term. Table 7 lists the regular expression operators. You place an operator immediately after term with no intervening space, except for the pipe (|) operator, which you place between two terms.

Community regular expressions are identical to the UNIX regular expressions. Both implement the extended (or modern) regular expressions as defined in POSIX 1003.2.

Table 7 on page 67 shows some of the operators you can use with community regular expressions.

When you define communities, you can do the following:

Do Not Advertise Communities to Neighbors on page 74

Configure the BGP Extended Communities Attribute on page 75

Examples: Configure BGP Extended Community Attribute on page 75

Examples: Define Communities to Use in Routing Policies

Create a community named *dunedin* and apply it in a policy statement:

```
[edit]
policy-options {
  community dunedin members [56:2379 23:46944];
  policy-statement from-dunedin {
    from community dunedin;
    then {
      metric 2;
      preference 100;
      next policy;
    }
  }
}
```

Remove the communities 65535:10 and 65535:11, or 65534:10 and 65534:11 from routes:

```
[edit]
policy-options {
  community my-as1-transit members [65535:10 65535:11];
  community my-as2-transit members [65534:10 65534:11];
  community my-wild members [65534:* 65535:*];
  policy-statement delete-communities {
    from {
      community [my-as1-transit my-as2-transit];
    }
    then {
      community delete my-wild;
    }
  }
}
```

Match the set of community members 5000, 5010, 5020, 5030, and so on up to 5090:

```
[edit]
policy-options {
  community customers members [1111:5..0]
  policy-statement advertise-customers {
    from community customers;
    then accept;
  }
}
```

Suppress the advertisement of all communities from all ASs. In the wildcard match `*:*`, the first asterisk matches all AS numbers and the second matches all community identifiers. To suppress the advertisement of all communities from a particular AS, define the community as community wild members `as-number:*`. To suppress a particular community from any AS, define the community as community wild members `*:community-value`.

```
[edit]
policy-options {
  community wild members " * : * ";
  policy-statement suppress-communities {
    then {
      community delete wild;
    }
  }
}
```

Reject routes that are longer than /19 in classical Class A space, /16 in classical Class B space, and /24 in classical Class C space:

```
[edit]
community auckland-accept members 666:1;
policy-statement drop-specific-routes {
  from {
    route-filter 0.0.0.0/1 upto /19
      community add auckland-accept
      next policy;
    route-filter 128.0.0.0/2 upto /16
      community add auckland-accept
      next policy;
    route-filter 0.0.0.0/0 upto /24
      community add auckland-accept
      next policy;
  }
  then reject;
}
```

Create policies to handle peer and customer communities:

```
[edit]
policy-options {
  community internal201 members 201:112;
  community internal202 members 202:112;
  community internal203 members 203:112;
  community internal204 members 204:112;
  community internal205 members 205:112;
  community peer201 members 201:666;
  community peer202 members 202:666;
  community peer203 members 203:666;
  community peer204 members 204:666;
  community peer205 members 205:666;
  community custpeer201 members 201:20;
  community custpeer202 members 202:20;
  community custpeer203 members 203:20;
  community custpeer204 members 204:20;
  community custpeer205 members 205:20;
  community prepend201-1 members 201:1;
  community prepend202-1 members 202:1;
  community prepend203-1 members 203:1;
  community prepend204-1 members 204:1;
  community prepend205-1 members 205:1;
  community prepend201-2 members 201:2;
  community prepend202-2 members 202:2;
  community prepend203-2 members 203:2;
  community prepend204-2 members 204:2;
  community prepend205-2 members 205:2;
  community prepend201-3 members 201:3;
  community prepend202-3 members 202:3;
  community prepend203-3 members 203:3;
  community prepend204-3 members 204:3;
  community prepend205-3 members 205:3;
  community lcl201-low members 201:80;
  community lcl202-low members 202:80;
  community lcl203-low members 203:80;
  community lcl204-low members 204:80;
  community lcl205-low members 205:80;
  community lcl201-high members 201:120;
  community lcl202-high members 202:120;
  community lcl203-high members 203:120;
```

```

community lcl204-high members 204:120;
community lcl205-high members 205:120;
policy-statement in-customer {
  term term1 {
    from {
      protocol bgp;
      community [lcl201-low lcl202-low lcl203-low lcl204-low lcl205-low];
    }
    then {
      local-preference 80;
      accept;
    }
  }
  term term2 {
    from {
      protocol bgp;
      community [lcl201-high lcl202-high lcl203-high lcl204-high lcl205-high];
    }
    then local-preference 120;
  }
  then next policy;
}
policy-statement out-customer {
  term term1 {
    from {
      protocol bgp;
      community [internal201 internal202 internal203 internal204 internal205];
    }
    then reject;
  }
  then next policy;
}
policy-statement in-peer {
  from protocol bgp;
  then {
    metric 10;
    community = peer201;
  }
}
policy-statement out-peer {
  term term1 {
    from {
      protocol bgp;
      community [prepend201-1 prepend202-1 prepend203-1 prepend204-1
        prepend205-1];
    }
    then as-path-prepend 201;
  }
  term term2 {
    from {
      protocol bgp;
      community [prepend201-2 prepend202-2 prepend203-2 prepend204-2
        prepend205-2];
    }
    then as-path-prepend "201 201";
  }
}

```

```

term term3 {
  from {
    protocol bgp;
    community [prepend201-3 prepend202-3 prepend203-3
              prepend204-3 prepend205-3];
  }
  then as-path-prepend "201 201 201";
}
term term4 {
  from {
    protocol bgp;
    community [peer201 peer202 peer203 peer204 peer205 custpeer201
              custpeer202 custpeer203 custpeer204 custpeer205 internal201
              internal202 internal203 internal204 internal205];
  }
  then reject;
}
then next policy;
}
}

```

Do Not Advertise Communities to Neighbors

By default, communities are sent to peer groups. To suppress the advertisement of communities to a neighbor, remove all communities. When the result of an export policy is an empty set of communities, the community attribute is not sent. To remove all communities, define a wildcard set of communities:

```

[edit policy-options]
community wild members "*:*";

```

Then in the policy statement, specify this community in the action:

```

[edit policy-options]
policy-statement policy-name {
  term term-name {
    then community delete wild;
  }
}

```

Configure the BGP Extended Communities Attribute

The BGP extended communities attribute provides a larger range (8 octets) for grouping or categorizing communities for applying routing decisions, such as acceptance, rejection, preference, or redistribution. The community attribute is only 4 octets.

To configure extended communities, include the community statement at the [edit policy-options] hierarchy level, specifying the type of community members as either target or origin. For examples of extended BGP communities, see “Examples: Configure BGP Extended Community Attribute” on page 75.

```
[edit policy-options]
  community name members [community-ids];
```

name—identifies one or more routers in the BGP extended community.

community-ids—identifies the type of extended community. The community-id includes the following fields:

Type of extended community—identifies either a target or origin community. The target community identifies the destination to which the route is going. The origin community identifies where the route originated.

Administrator—depending on the type of extended community, either an Autonomous System (AS) number or an IPv4 address prefix.

Assigned Number—identifies the local provider.

For more information about BGP extended communities, see *BGP Extended Communities Attribute*, Internet draft-ramachandra-bgp-ext-communities-04.txt.



Note

Regular expressions are not supported for extended communities.

Examples: Configure BGP Extended Community Attribute

Configure a target community with an administrative field of 10458 and an assigned number of 20:

```
[edit]
  policy-options {
    community test-a members [target:10458:20];
  }
```

Configure a target community with the administrative field of 1.1.1.1 and an assigned number of 20:

```
[edit]
  policy-options {
    community test-a members [target:1.1.1.1:20];
  }
```

Configure an origin community with the administrative field of 1.1.1.1 and an assigned number of 20:

```
[edit]
policy-options {
  community test-a members [origin:1.1.1.1:20];
}
```



