Chapter 33
Route Aggregation and Generation

33.1 Route Aggregation Overview

Route aggregation is a method of generating a more general summary route, given the presence of a more specific route. It is used, for example, at an autonomous system border to generate a route to a network to be advertised via BGP, given the presence of one or more subnets of that network learned via RIP. No aggregation is performed unless explicitly requested in an **`aggregate`** statement.

Route aggregation is also used by regional and national networks to reduce the amount of routing information passed. With careful allocation of network addresses to clients, regional networks can announce one route to regional networks instead of hundreds.

Aggregate routes are not actually used for packet forwarding by the originator of the aggregate route, only by the receiver (if it wishes). A router is supposed to respond with an ICMP network unreachable message if the router receives a packet that does not match one of the component routes that led to the generation of an aggregate route. This message is to prevent packets for unknown component routes from following a default route into another network, where they would be forwarded back to the border router, and around and around again and again, until their TTL expired. Sending an unreachable message for a missing piece of an aggregate is only possible on systems with support for reject routes.

A slight variation of aggregation is the generation of a route based on the existence of certain conditions. This is sometimes known as the “route of last resort.” This route inherits the next hops and AS path from the contributor specified with the lowest (most favorable) preference. The most common usage for this is to generate a default based on the presence of a route from a peer on a neighboring backbone.

33.1.1 Aggregate and Generate Inheritance

The following parameters can be specified at multiple places within a given **`aggregate`** or **`generate`** statement:

- **`restrict`**
- **`preference`**

In the case of **`restrict`**, placing it on an **`aggregate/generate`** source statement restricts inclusion of all routes from the specified protocol in the aggregate. Including it on a *`route_filter`* excludes all routes that match that route filter.

In the case of aggregation preference, the most specific instance of a **`preference`** value is assigned to a route. Thus, any **`preference`** value specified on an **`aggregate/generate`** statement is used only if neither the **`aggregate/generate`** source statement nor the *`route_filter`* that matches a route specifies a value. The value of the most specific matching *`route_filter`* is used. If neither the **`aggregate`**/**`generate`** statement, the matching **`aggregate`**/**`generate`** source statement, nor the matching *`route_filter`* specifies a preference value, then the default aggregate preference value of 130 is used.

33.2 Aggregation and Generation Syntax

```
aggregate ( [ inet6 ] default |
  network ( mask mask | masklen masklen | / masklen ) )
    [ preference preference_value ]
    [ bgp ]
    [ brief ]
    [ [ toribs ] ( [ unicast | multicast | unicast multicast ) ]
    {
        aggregate_list
    };
generate ( [ inet6 ] default |
  network ( mask mask | masklen masklen | / masklen ) )
    [ preference preference_value ]
    [ [ toribs ] ( unicast | multicast | unicast multicast ) ]
    [ noinstall ]
    {
        aggregate_list
    };
```

aggregate_list:

Aggregate lists are composed of one or more of the following:

```
    proto bgp
        [ ( as ASN ) | ( aspath aspath-regular-expression
        origin ( any | igp | egp | incomplete ) ) ]
        [ comm { communities_list } ]
        [ ext-comm { extended_communities_list } ]
        [ preference preference ]
        { [ route_filter [ preference preference | restrict ] ; ] } ;
    proto bgp
        [ ( as ASN ) | ( aspath aspath-regular-expression
        origin ( any | igp | egp | incomplete ) ) ]
        restrict  ;

    proto rip
        [ tag tagvalue ]
    restrict ;
proto rip
    [ tag tagvalue ]
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ripng
    [ tag tagvalue ]
    restrict ;
proto ripng
    [ tag tagvalue ]
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ospf [ tag tagvalue ] restrict ;
proto ospf [ tag tagvalue ] [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto ospfase [ tag tagvalue ] restrict ;
proto ospfase [ tag tagvalue ] [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto direct restrict ;
proto direct
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto static restrict ;
proto static
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto kernel restrict ;
proto kernel
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto isis restrict ;
proto isis
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto aggregate restrict ;
proto aggregate
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;

proto all restrict ;
proto all
    [ preference preference ]
    { [ route_filter [ preference preference | restrict ] ; ] } ;
```

More detailed descriptions of these commands can be found on page 667 of the *Command Reference Guide*.

Routes that match the route filters are called “contributing” routes. They are ordered on the list of contributing routes for a given aggregate route according to the aggregation preference value that applies to them, with the lowest (best) preference values coming first. The aggregation preference value can be specified on the **`aggregate`** or **`generate`** statement, any **`aggregate`** source statement, or any *`route_filters`*. In addition to being used to order contributing routes, the aggregation preference value associated with the first contributor on this ordered list is used as the route preference value for the aggregate route itself.

When ordering contributing routes on the contributor list, if two contributors have the same aggregation preference value, then the contributor with the lowest route preference is ordered before the other contributors with the same aggregation preference value.

In the case of **`generate`**, in addition to using the aggregation preference value of the first contributor in the list as the aggregate route’s preference, the nexthops used for the aggregate are taken to be those of the first contributor in the list.

The aspath for an aggregate is formed by combining the aspath of each contributor. If BGP rules are configured for aggregation, the contributor order can impact the MED and nexthop values in the combined aspath generated for the aggregate. This is due to the fact that the values of MED and nexthop from the first valid BGP contributor route encountered while traversing the contributor list in order are used in the combined aspath.

A route can contribute only to an aggregate route that is more general (less specific) than itself; it must match the aggregate under its mask. Any given route can contribute only to one aggregate route, which will be the most specific configured, but an aggregate route may contribute to a more general aggregate.

Note that a route is only considered as a potential contributor to aggregate routes for which it is a more specific route. Therefore, a filter of **`all`** only pertains to more specific routes of the aggregate. Currently, specifying a *`network`* type filter for a network that is not a more specific instance of the aggregate to which it pertains is not treated as an error even though it will never match any contributors.

33.3 Exporting Generated vs. Aggregated Routes

If you create an aggregate and export it, the result achieved is that which is expected: for the general aggregate, a loopback (reject) route is installed in the kernel, and the route is advertised with the aggregating router as the next hop.

Consider the following topology

```
    ---------         ---------       ---------
    | RTR A |         | RTR B |       | RTR C |
    ----+----         ----+----       ----+----
        |                 |               |
--------+-----------------+---------------+------
```

wherein routers A and B are OSPF peers and routers B and C are BGP peers. Let router A advertise a static route to 223.50. Let router B have the following configuration

```
aggregate 223 masklen 8 {
    proto ospfase {
         223 masklen 8 refines;
    };
}

export bgp peeras 123 {
    proto aggregate {
        all;
    };
};
```

Router B will install in its kernel

```
     223.50 gw RTR A
     223/8 gw 127.0.0.1 reject
```

and will advertise to Router C a route to 223/8 gw Router B. On the other hand, if the same configuration is used but with `generate` instead of `aggregate`, so we have

```
generate 223 masklen 8 {
    proto ospfase {
        223 masklen 8 refines ;
    };
};
```

then two things happen. Router B, instead of installing a reject route for portions of the aggregate that do not have specific matches, will instead install the following routes in the kernel:

```
223/8 gw RTR A
223.50 gw RTR A
```

The second difference is that the aggregate route advertised to router C will be 223/8 gw router A.

You can also configure an aggregate 10/7 on behalf of another client AS (for example, AS 65003):

```
aggregate 10.0.0.0 masklen 7 {
    proto bgp as 65003{
        10.0.0.0 masklen 8;
        11.0.0.0 masklen 8;
    };
};
```

33.4 Aggregating into Unicast and Multicast RIBs

RIBs need not be specified for aggregate routes. By default, an aggregate applies to all RIBs to which any contributing route applies. For example, an aggregate applies to the Unicast RIB if and only if any contributing route applies to the Unicast RIB.

Examples

```
aggregate 10.0.0.0 masklen 8 {
    proto static {
        10.0.0.0 masklen 8 refines;
      };
};
```

If any static route in the Unicast RIB matches the route filter, the aggregate will exist in the Unicast RIB. Likewise, for the Multicast RIB.

RIB limits can, however, be specified. By default, the limit is all RIBs (i.e., all RIBs to which any contributing route applies). This default can be overridden with a more specific limit, as in the example below:

```
aggregate 10.0.0.0 masklen 8 unicast {
    proto static {
        10.0.0.0 masklen 8 refines;
    };
};
```

The above aggregate applies only to the Unicast RIB (and only if a contributing route is in the Unicast RIB). Contributing routes in other RIBs are ignored.