RTGWG Y. Qu
Internet-Draft Futurewei
Intended status: Standards Track J. Tantsura
Expires: July 5, 2020 Apstra
A. Lindem
Cisco
X. Liu
Volta Networks
January 2, 2020

A YANG Data Model for Routing Policy Management
draft-ietf-rtgwg-policy-model-08

Abstract

This document defines a YANG data model for configuring and managing routing policies in a vendor-neutral way and based on actual operational practice. The model provides a generic policy framework which can be augmented with protocol-specific policy configuration.

Table of Contents

1. Introduction

This document describes a YANG [RFC6020] [RFC7950] data model for routing policy configuration based on operational usage and best practices in a variety of service provider networks. The model is intended to be vendor-neutral, in order to allow operators to manage policy configuration in a consistent, intuitive way in heterogeneous environments with routers supplied by multiple vendors.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342].

1.1. Goals and approach

This model does not aim to be feature complete -- it is a subset of the policy configuration parameters available in a variety of vendor implementations, but supports widely used constructs for managing how routes are imported, exported, and modified across different routing protocols. The model development approach has been to examine actual policy configurations in use across a number of operator networks. Hence the focus is on enabling policy configuration capabilities and structure that are in wide use.

Despite the differences in details of policy expressions and conventions in various vendor implementations, the model reflects the observation that a relatively simple condition-action approach can be readily mapped to several existing vendor implementations, and also gives operators an intuitive and straightforward way to express policy without sacrificing flexibility. A side affect of this design decision is that legacy methods for expressing policies are not considered. Such methods could be added as an augmentation to the model if needed.

Consistent with the goal to produce a data model that is vendor neutral, only policy expressions that are deemed to be widely available in existing major implementations are included in the model. Those configuration items that are only available from a single implementation are omitted from the model with the expectation they will be available in separate vendor-provided modules that augment the current model.

2. Terminology and Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined in [RFC8342]:

- client
- server
- configuration
- system state
- operational state

o intended configuration

The following terms are defined in [RFC7950]:

o action

o augment

o container

o container with presence

o data model

o data node

o feature

o leaf

o list

o mandatory node

o module

o schema tree

o RPC (Remote Procedure Call) operation

2.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340].

2.2. Prefixes in Data Node Names

In this document, names of data nodes, actions, and other data model objects are often used without a prefix, as long as it is clear from the context in which YANG module each name is defined. Otherwise, names are prefixed using the standard prefix associated with the corresponding YANG module, as shown in Table 1.

Prefix	YANG module	Reference
if	ietf-interfaces	[RFC8343]
rt	ietf-routing	[RFC8349]
yang	ietf-yang-types	[RFC6991]
inet	ietf-inet-types	[RFC6991]
if-ext	ietf-if-extensions	[I-D.ietf-netmod-intf-ext-yang]
if-l3-vlan	ietf-if-l3-vlan	[I-D.ietf-netmod-sub-intf-vlan-model]

Table 1: Prefixes and Corresponding YANG Modules

3. Model overview

The routing policy module has three main parts:

- o A generic framework to express policies as sets of related conditions and actions. This includes match sets and actions that are useful across many routing protocols.

- o A structure that allows routing protocol models to add protocol-specific policy conditions and actions though YANG augmentations. There is a complete example of this for BGP [RFC4271] policies in the proposed vendor-neutral BGP data model [I-D.ietf-idr-bgp-model].

- o A reusable grouping for attaching import and export rules in the context of routing configuration for different protocols, VRFs, etc. This also enables creation of policy chains and expressing default policy behavior.

The module makes use of the standard Internet types, such as IP addresses, autonomous system numbers, etc., defined in RFC 6991 [RFC6991].

4. Route policy expression

Policies are expressed as a sequence of top-level policy definitions each of which consists of a sequence of policy statements. Policy statements in turn consist of simple condition-action tuples.

Conditions may include multiple match or comparison operations, and similarly, actions may effect multiple changes to route attributes, or indicate a final disposition of accepting or rejecting the route. This structure is shown below.

```
+--rw routing-policy
   +--rw policy-definitions
      +--rw policy-definition* [name]
         +--rw name          string
         +--rw statements
            +--rw statement* [name]
               +--rw name          string
               +--rw conditions
               |     ...
               +--rw actions
                     ...
```

4.1. Defined sets for policy matching

The models provides a set of generic sets that can be used for matching in policy conditions. These sets are applicable for route selection across multiple routing protocols. They may be further augmented by protocol-specific models which have their own defined sets. The supported defined sets include:

- prefix sets - define a set of IP prefixes, each with an associated CIDR netmask range (or exact length)

- neighbor sets - define a set of neighboring nodes by their IP addresses. These sets are used for selecting routes based on the neighbors advertising the routes.

- tag set - define a set of generic tag values that can be used in matches for filtering routes

The model structure for defined sets is shown below.

```
+--rw routing-policy
   +--rw defined-sets
   |  +--rw prefix-sets
   |  |  +--rw prefix-set* [name]
   |  |     +--rw name        string
   |  |     +--rw mode?       enumeration
   |  |     +--rw prefixes
   |  |        +--rw prefix-list* [ip-prefix masklength-lower
   |  |                            masklength-upper]
   |  |           +--rw ip-prefix           inet:ip-prefix
   |  |           +--rw masklength-lower    uint8
   |  |           +--rw masklength-upper    uint8
   |  +--rw neighbor-sets
   |  |  +--rw neighbor-set* [name]
   |  |     +--rw name       string
   |  |     +--rw address*   inet:ip-address
   |  +--rw tag-sets
   |     +--rw tag-set* [name]
   |        +--rw name         string
   |        +--rw tag-value*   tag-type
```

4.2. Policy conditions

Policy statements consist of a set of conditions and actions (either of which may be empty). Conditions are used to match route attributes against a defined set (e.g., a prefix set), or to compare attributes against a specific value.

Match conditions may be further modified using the match-set-options configuration which allows operators to change the behavior of a match. Three options are supported:

- ALL - match is true only if the given value matches all members of the set.

- ANY - match is true if the given value matches any member of the set.

- INVERT - match is true if the given value does not match any member of the given set.

Not all options are appropriate for matching against all defined sets (e.g., match ALL in a prefix set does not make sense). In the model, a restricted set of match options is used where applicable.

Comparison conditions may similarly use options to change how route attributes should be tested, e.g., for equality or inequality, against a given value.

While most policy conditions will be added by individual routing protocol models via augmentation, this routing policy model includes several generic match conditions and also the ability to test which protocol or mechanism installed a route (e.g., BGP, IGP, static, etc.). The conditions included in the model are shown below.

```
+--rw routing-policy
  +--rw policy-definitions
      +--rw policy-definition* [name]
         +--rw name          string
         +--rw statements
            +--rw statement* [name]
               +--rw conditions
               |  +--rw call-policy?
               |  +--rw install-protocol-eq?
               |  +--rw match-interface
               |  |  +--rw interface?
               |  |  +--rw subinterface?
               |  +--rw match-prefix-set
               |  |  +--rw prefix-set?
               |  |  +--rw match-set-options?
               |  +--rw match-neighbor-set
               |  |  +--rw neighbor-set?
               |  +--rw match-tag-set
               |     +--rw tag-set?
               |     +--rw match-set-options?
```

4.3. Policy actions

When policy conditions are satisfied, policy actions are used to set various attributes of the route being processed, or to indicate the final disposition of the route, i.e., accept or reject.

Similar to policy conditions, the routing policy model includes generic actions in addition to the basic route disposition actions. These are shown below.

```
       +--rw routing-policy
         +--rw policy-definitions
            +--rw policy-definition* [name]
               +--rw statements
                  +--rw statement* [name]
                     +--rw actions
                        +--rw policy-result?    policy-result-type
                        +--rw set-metric?       uint32
                        +--rw set-preference?   uint16
```

4.4. Policy subroutines

Policy 'subroutines' (or nested policies) are supported by allowing policy statement conditions to reference other policy definitions using the call-policy configuration. Called policies apply their conditions and actions before returning to the calling policy statement and resuming evaluation. The outcome of the called policy affects the evaluation of the calling policy. If the called policy results in an accept-route, then the subroutine returns an effective boolean true value to the calling policy. For the calling policy, this is equivalent to a condition statement evaluating to a true value and evaluation of the policy continues (see Section 5). Note that the called policy may also modify attributes of the route in its action statements. Similarly, a reject-route action returns false and the calling policy evaluation will be affected accordingly. When the end of the subroutine policy chain is reached, the default route disposition action is returned (i.e., boolean false for reject-route unless an alternate default action is specified for the chain). Consequently, a subroutine cannot explicitly accept or reject a route. Rather it merely provides an indication that 'call-policy' condition returns boolean true or false indicating whether or not the condition matches. Route acceptance or rejection is solely determined by the top-level policy.

Note that the called policy may itself call other policies (subject to implementation limitations). The model does not prescribe a nesting depth because this varies among implementations. For example, some major implementation may only support a single level of subroutine recursion. As with any routing policy construction, care must be taken with nested policies to ensure that the effective return value results in the intended behavior. Nested policies are a convenience in many routing policy constructions but creating policies nested beyond a small number of levels (e.g., 2-3) should be discouraged.

5. Policy evaluation

Evaluation of each policy definition proceeds by evaluating its corresponding individual policy statements in order. When all the condition statements in a policy statement are satisfied, the corresponding action statements are executed. If the actions include either accept-route or reject-route actions, evaluation of the current policy definition stops, and no further policy definitions in the chain are evaluated.

If the conditions are not satisfied, then evaluation proceeds to the next policy statement. If none of the policy statement conditions are satisfied, then evaluation of the current policy definition stops, and the next policy definition in the chain is evaluated. When the end of the policy chain is reached, the default route disposition action is performed (i.e., reject-route unless an alternate default action is specified for the chain).

Note that the route's pre-policy attributes are always used for testing policy statement conditions. In other words, if actions modify the policy application specific attributes, those modifications are not used for policy statement conditions.

6. Applying routing policy

Routing policy is applied by defining and attaching policy chains in various routing contexts. Policy chains are sequences of policy definitions (described in Section 4) that have an associated direction (import or export) with respect to the routing context in which they are defined. The routing policy model defines an apply-policy grouping that can be imported and used by other models. As shown below, it allows definition of import and export policy chains, as well as specifying the default route disposition to be used when no policy definition in the chain results in a final decision.

```
      +--rw apply-policy
      |  +--rw import-policy*
      |  +--rw default-import-policy?   default-policy-type
      |  +--rw export-policy*
      |  +--rw default-export-policy?   default-policy-type
```

The default policy defined by the model is to reject the route for both import and export policies.

7. Routing protocol-specific policies

Routing models that require the ability to apply routing policy may augment the routing policy model with protocol or other specific policy configuration. The routing policy model assumes that additional defined sets, conditions, and actions may all be added by other models.

An example of this is shown below, in which the BGP configuration model in [I-D.ietf-idr-bgp-model] adds new defined sets to match on community values or AS paths. The model similarly augments BGP-specific conditions and actions in the corresponding sections of the routing policy model.

```
+--rw routing-policy
   +--rw defined-sets
   |  +--rw prefix-sets
   |  |  +--rw prefix-set* [name]
   |  |     +--rw name        string
   |  |     +--rw mode?       enumeration
   |  |     +--rw prefixes
   |  |        +--rw prefix-list* [ip-prefix masklength-lower
   |  |                            masklength-upper]
   |  |           +--rw ip-prefix           inet:ip-prefix
   |  |           +--rw masklength-lower    uint8
   |  |           +--rw masklength-upper    uint8
   |  +--rw neighbor-sets
   |  |  +--rw neighbor-set* [name]
   |  |     +--rw name       string
   |  |     +--rw address*   inet:ip-address
   |  +--rw tag-sets
   |  |  +--rw tag-set* [name]
   |  |     +--rw name         string
   |  |     +--rw tag-value*   tag-type
   |  +--rw bgp-pol:bgp-defined-sets
   |     +--rw bgp-pol:community-sets
   |     |  +--rw bgp-pol:community-set* [community-set-name]
   |     |     +--rw bgp-pol:community-set-name    string
   |     |     +--rw bgp-pol:community-member*     union
   |     +--rw bgp-pol:ext-community-sets
   |     |  +--rw bgp-pol:ext-community-set* [ext-community-set-name]
   |     |     +--rw bgp-pol:ext-community-set-name    string
   |     |     +--rw bgp-pol:ext-community-member*     union
   |     +--rw bgp-pol:as-path-sets
   |        +--rw bgp-pol:as-path-set* [as-path-set-name]
   |           +--rw bgp-pol:as-path-set-name      string
   |           +--rw bgp-pol:as-path-set-member*   string
   +--rw policy-definitions
```

```
         +--rw policy-definition* [name]
            +--rw name          string
            +--rw statements
               +--rw statement* [name]
                  +--rw name          string
                  +--rw conditions
                  |  +--rw call-policy?
                  |  +--rw source-protocol?          identityref
                  |  +--rw match-interface
                  |  |  +--rw interface?
                  |  |  +--rw subinterface?
                  |  +--rw match-prefix-set
                  |  |  +--rw prefix-set?
                  |  |  +--rw match-set-options? match-set-options-type
                  |  +--rw match-neighbor-set
                  |  |  +--rw neighbor-set?
                  |  +--rw match-tag-set
                  |  |  +--rw tag-set?
                  |  |  +--rw match-set-options? match-set-options-type
                  |  +--rw bgp-pol:bgp-conditions
                  |     +--rw bgp-pol:med-eq?      uint32
                  |     +--rw bgp-pol:origin-eq?
                  |           bgp-types:bgp-origin-attr-type
                  |     +--rw bgp-pol:next-hop-in*
                  |           inet:ip-address-no-zone
                  |     +--rw bgp-pol:afi-safi-in*   identityref
                  |     +--rw bgp-pol:local-pref-eq? uint32
                  |     +--rw bgp-pol:route-type?    enumeration
                  |     +--rw bgp-pol:community-count
                  |     +--rw bgp-pol:as-path-length
                  |     +--rw bgp-pol:match-community-set
                  |     |  +--rw bgp-pol:community-set?
                  |     |  +--rw bgp-pol:match-set-options?
                  |              match-set-options-type
                  |     +--rw bgp-pol:match-ext-community-set
                  |     |  +--rw bgp-pol:ext-community-set?
                  |     |  +--rw bgp-pol:match-set-options?
                  |     |        match-set-options-type
                  |     +--rw bgp-pol:match-as-path-set
                  |        +--rw bgp-pol:as-path-set?
                  |        +--rw bgp-pol:match-set-options?
                  |              match-set-options-type
                  +--rw actions
                     +--rw policy-result?         policy-result-type
                     +--rw set-metric?            uint32
                     +--rw set-preference?        uint16
                     +--rw bgp-pol:bgp-actions
                        +--rw bgp-pol:set-route-origin?
```

```
            bgp-types:bgp-origin-attr-type
      +--rw bgp-pol:set-local-pref?       uint32
      +--rw bgp-pol:set-next-hop?   bgp-next-hop-type
      +--rw bgp-pol:set-med?        bgp-set-med-type
      +--rw bgp-pol:set-as-path-prepend
      |  +--rw bgp-pol:repeat-n?   uint8
      +--rw bgp-pol:set-community
      |  +--rw bgp-pol:method?      enumeration
      |  +--rw bgp-pol:options?
      |          bgp-set-community-option-type
      |  +--rw bgp-pol:inline
      |  |  +--rw bgp-pol:communities*   union
      |  +--rw bgp-pol:reference
      |     +--rw bgp-pol:community-set-ref?
      +--rw bgp-pol:set-ext-community
         +--rw bgp-pol:method?       enumeration
         +--rw bgp-pol:options?
                 bgp-set-community-option-type
         +--rw bgp-pol:inline
         |  +--rw bgp-pol:communities*   union
         +--rw bgp-pol:reference
            +--rw bgp-pol:ext-community-set-ref?
```

8. Security Considerations

Routing policy configuration has a significant impact on network operations, and, as such, any related model carries potential security risks.

YANG data models are generally designed to be used with the NETCONF protocol over an SSH transport. This provides an authenticated and secure channel over which to transfer configuration and operational data. Note that use of alternate transport or data encoding (e.g., JSON over HTTPS) would require similar mechanisms for authenticating and securing access to configuration data.

Most of the data elements in the policy model could be considered sensitive from a security standpoint. Unauthorized access or invalid data could cause major disruption.

9. IANA Considerations

This YANG data model and the component modules currently use a temporary ad-hoc namespace. If and when it is placed on redirected for the standards track, an appropriate namespace URI will be registered in the IETF XML Registry" [RFC3688]. The routing policy YANG modules will be registered in the "YANG Module Names" registry [RFC6020].

10. YANG modules

The routing policy model is described by the YANG modules in the sections below.

10.1. Routing policy model

```
<CODE BEGINS> file "ietf-routing-policy@2020-01-02.yang"
module ietf-routing-policy {

  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-routing-policy";
  prefix rt-pol;

  import ietf-inet-types {
    prefix "inet";
  }

  import ietf-yang-types {
    prefix "yang";
  }

  import ietf-interfaces {
    prefix "if";
  }

  import ietf-routing {
    prefix "rt";
  }

  import ietf-if-extensions {
    prefix if-ext;
  }

  import ietf-if-l3-vlan {
    prefix "if-l3-vlan";
  }


 organization
    "IETF RTGWG - Routing Area Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/rtgwg/>
     WG List:  <mailto:rtgwg@ietf.org>


     Editor:   Yingzhen Qu
               <mailto:yingzhen.qu@futurewei.com>
```

```
                Jeff Tantsura
                <mailto:jefftant.ietf@gmail.com>
                Acee Lindem
                <mailto:acee@cisco.com>
                Xufeng Liu
                <mailto:xufeng_liu@jabil.com>
                Anees Shaikh
                <mailto:aashaikh@google.com>";

     description
       "This module describes a YANG model for routing policy
        configuration. It is a limited subset of all of the policy
        configuration parameters available in the variety of vendor
        implementations, but supports widely used constructs for
        managing how routes are imported, exported, and modified across
        different routing protocols.  This module is intended to be
        used in conjunction with routing protocol configuration modules
        (e.g., BGP) defined in other models.

        Copyright (c) 2020 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject to
        the license terms contained in, the Simplified BSD License set
        forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
        for full legal notices.

        Route policy expression:

        Policies are expressed as a set of top-level policy
        definitions, each of which consists of a sequence of policy
        statements. Policy statements consist of simple
        condition-action tuples. Conditions may include mutiple match
        or comparison operations, and similarly actions may be
        multitude of changes to route attributes or a final disposition
        of accepting or rejecting the route.

        Route policy evaluation:

        Policy definitions are referenced in routing protocol
        configurations using import and export configuration
        statements. The arguments are members of an ordered list of
```

```
         named policy definitions which comprise a policy chain, and
         optionally, an explicit default policy action (i.e., reject
         or accept).

         Evaluation of each policy definition proceeds by evaluating its
         corresponding individual policy statements in order.  When a
         condition statement in a policy statement is satisfied, the
         corresponding action statement is executed.  If the action
         statement has either accept-route or reject-route actions,
         policy evaluation of the current policy definition stops, and
         no further policy definitions in the chain are evaluated.

         If the condition is not satisfied, then evaluation proceeds to
         the next policy statement.  If none of the policy statement
         conditions are satisfied, then evaluation of the current policy
         definition stops, and the next policy definition in the chain
         is evaluated.  When the end of the policy chain is reached, the
         default route disposition action is performed (i.e.,
         reject-route unless an alternate default action is specified
         for the chain).

         Policy 'subroutines' (or nested policies) are supported by
         allowing policy statement conditions to reference another
         policy definition which applies conditions and actions from
         the referenced policy before returning to the calling policy
         statement and resuming evaluation.  If the called policy
         results in an accept-route (either explicit or by default),
         then the subroutine returns an effective true value to the
         calling policy.  Similarly, a reject-route action returns
         false.  If the subroutine returns true, the calling policy
         continues to evaluate the remaining conditions (using a
         modified route if the subroutine performed any changes to the
         route).";


      revision "2020-01-02" {
        description
          "Initial revision.";
        reference
         "RFC XXXX: Routing Policy Configuration Model for Service
          Provider Networks";
      }


      // typedef statements

      typedef default-policy-type {
        // this typedef retained for name compatibiity with default
```

```
        // import and export policy
        type enumeration {
          enum accept-route {
            description
              "Default policy to accept the route";
          }
          enum reject-route {
            description
              "Default policy to reject the route";
          }
        }
        description
          "Type used to specify route disposition in
           a policy chain";
      }

      typedef policy-result-type {
        type enumeration {
          enum accept-route {
            description "Policy accepts the route";
          }
          enum reject-route {
            description "Policy rejects the route";
          }
        }
        description
          "Type used to specify route disposition in
           a policy chain";
      }

      typedef tag-type {
        type union {
          type uint32;
          type yang:hex-string;
        }
        description "Type for expressing route tags on a local system,
           including IS-IS and OSPF; may be expressed as either decimal
           or hexadecimal integer";
        reference
          "RFC 2178 - OSPF Version 2
           RFC 5130 - A Policy Control Mechanism in IS-IS Using
                      Administrative Tags";
      }

      typedef match-set-options-type {
        type enumeration {
          enum any {
            description "Match is true if given value matches any member
```

```
              of the defined set";
         }
         enum all {
           description "Match is true if given value matches all
              members of the defined set";
         }
         enum invert {
           description "Match is true if given value does not match any
              member of the defined set";
         }
       }
       default any;
       description
         "Options that govern the behavior of a match statement.  The
          default behavior is any, i.e., the given value matches any
          of the members of the defined set";
     }


     // grouping statements

     grouping prefix-set {
       description
         "Configuration data for prefix sets used in policy
          definitions.";

       leaf name {
         type string;
         description
           "Name of the prefix set -- this is used as a label to
            reference the set in match conditions";
       }

       leaf mode {
         type enumeration {
           enum ipv4 {
             description
               "Prefix set contains IPv4 prefixes only";
           }
           enum ipv6 {
             description
               "Prefix set contains IPv6 prefixes only";
           }
           enum mixed {
             description
               "Prefix set contains mixed IPv4 and IPv6 prefixes";
           }
         }
```

```
         description
           "Indicates the mode of the prefix set, in terms of which
            address families (IPv4, IPv6, or both) are present.  The
            mode provides a hint, but the device must validate that all
            prefixes are of the indicated type, and is expected to
            reject the configuration if there is a discrepancy.  The
            MIXED mode may not be supported on devices that require
            prefix sets to be of only one address family.";
       }

     }

     grouping prefix-set-top {
       description
         "Top-level data definitions for a list of IPv4 or IPv6
          prefixes which are matched as part of a policy";

       container prefix-sets {
         description
           "Enclosing container ";

         list prefix-set {
           key "name";
           description
             "List of the defined prefix sets";

           uses prefix-set;

           uses prefix-top;
         }
       }
     }

     grouping prefix {
       description
         "Configuration data for a prefix definition";

       leaf ip-prefix {
         type inet:ip-prefix;
         mandatory true;
         description
           "The prefix member in CIDR notation -- while the
            prefix may be either IPv4 or IPv6, most
            implementations require all members of the prefix set
            to be the same address family.  Mixing address types in
            the same prefix set is likely to cause an error.";
       }
```

```
       leaf masklength-lower {
         type uint8;
         description
           "Masklength range lower bound.";
       }
       leaf masklength-upper {
         type uint8 {
           range "1..128";
         }
         must "../masklength-upper >= ../masklength-lower" {
           error-message "The upper bound should not be less"
                       + "than lower bound.";
         }
         description
           "Masklength range upper bound.

            The combination of masklength-lower and masklength-upper
            define a range for the mask length, or single 'exact'
            length if masklength-lower and masklenght-upper are equal.

            Example: 10.3.192.0/21 through 10.3.192.0/24 would be
            expressed as prefix: 10.3.192.0/21,
                         masklength-lower=21,
                         masklength-upper=24

            Example: 10.3.192.0/21 (an exact match) would be
            expressed as prefix: 10.3.192.0/21,
                         masklength-lower=21,
                         masklength-upper=21";
       }
     }

     grouping prefix-top {
       description
         "Top-level grouping for prefixes in a prefix list";

       container prefixes {
         description
           "Enclosing container for the list of prefixes in a policy
            prefix list";

         list prefix-list {
           key "ip-prefix masklength-lower masklength-upper";
           description
             "List of prefixes in the prefix set";

           uses prefix;
         }
```

```
       }
     }

     grouping neighbor-set {
       description
         "This grouping provides neighbor set definitions";

       leaf name {
         type string;
         description
             "Name of the neighbor set -- this is used as a label
              to reference the set in match conditions";
       }

       leaf-list address {
         type inet:ip-address;
         description
           "List of IP addresses in the neighbor set";
       }
     }

     grouping neighbor-set-top {
       description
         "Top-level data definition for a list of IPv4 or IPv6
          neighbors which can be matched in a routing policy";

       container neighbor-sets {
         description
           "Enclosing container for the list of neighbor set
            definitions";

         list neighbor-set {
           key "name";
           description
             "List of defined neighbor sets for use in policies.";

           uses neighbor-set;
         }
       }
     }

     grouping tag-set {
       description
         "This grouping provides tag set definitions.";

       leaf name {
         type string;
         description
```

```
          "Name of the tag set -- this is used as a label to reference
           the set in match conditions";
      }

      leaf-list tag-value {
        type tag-type;
        description
          "Value of the tag set member";
      }
    }

    grouping tag-set-top {
      description
        "Top-level data definitions for a list of tags which can
         be matched in policies";

      container tag-sets {
        description
          "Enclosing container for the list of tag sets.";

        list tag-set {
          key "name";
          description
            "List of tag set definitions.";

          uses tag-set;

        }
      }
    }


    grouping match-set-options-group {
      description
        "Grouping containing options relating to how a particular set
         should be matched";

      leaf match-set-options {
        type match-set-options-type;
        description
          "Optional parameter that governs the behavior of the
           match operation";
      }
    }

    grouping match-set-options-restricted-group {
      description
        "Grouping for a restricted set of match operation modifiers";
```

```
       leaf match-set-options {
         type match-set-options-type {
           enum any {
             description "Match is true if given value matches any
                member of the defined set";
           }
           enum invert {
             description "Match is true if given value does not match
                any member of the defined set";
           }
         }
         description
           "Optional parameter that governs the behavior of the
            match operation.  This leaf only supports matching on ANY
            member of the set or inverting the match.  Matching on ALL
            is not supported";
       }
     }



     grouping match-interface-condition {
       description
         "This grouping provides interface match condition";

       container match-interface {
         leaf interface {
           type leafref {
             path "/if:interfaces/if:interface/if:name";
           }
           description
             "Reference to a base interface.  If a reference to a
              subinterface is required, this leaf must be specified
              to indicate the base interface.";
         }
         leaf subinterface {
           type leafref {
             path "/if:interfaces/if:interface/if-ext:encapsulation"
                + "/if-l3-vlan:dot1q-vlan"
                + "/if-l3-vlan:outer-tag/if-l3-vlan:vlan-id";
           }
           description
             "Reference to a subinterface -- this requires the base
              interface to be specified using the interface leaf in
              this container.  If only a reference to a base interface
              is requuired, this leaf should not be set.";
         }
```

```
        description
          "Container for interface match conditions";
      }
    }

    grouping prefix-set-condition {
      description
        "This grouping provides prefix-set conditions";

      container match-prefix-set {
        leaf prefix-set {
          type leafref {
            path "../../../../../../../defined-sets/" +
              "prefix-sets/prefix-set/name";
          }
          description "References a defined prefix set";
        }
        uses match-set-options-restricted-group;

        description
          "Match a referenced prefix-set according to the logic
           defined in the match-set-options leaf";
      }
    }

    grouping neighbor-set-condition {
      description
        "This grouping provides neighbor-set conditions";

      container match-neighbor-set {
        leaf neighbor-set {
          type leafref {
            path "../../../../../../../defined-sets/neighbor-sets/" +
            "neighbor-set/name";
            require-instance true;
          }
          description "References a defined neighbor set";
        }

        description
          "Match a referenced neighbor set according to the logic
           defined in the match-set-options-leaf";
      }
    }

    grouping tag-set-condition {
      description
        "This grouping provides tag-set conditions";
```

```
       container match-tag-set {
         leaf tag-set {
           type leafref {
             path  "../../../../../../../defined-sets/tag-sets" +
             "/tag-set/name";
             require-instance true;
           }
           description "References a defined tag set";
         }
         uses match-set-options-restricted-group;

         description
           "Match a referenced tag set according to the logic defined
            in the match-options-set leaf";
       }
     }

     grouping generic-conditions {
       description "Condition statement definitions for checking
          membership in a generic defined set";

       uses match-interface-condition;
       uses prefix-set-condition;
       uses neighbor-set-condition;
       uses tag-set-condition;

     }

     grouping policy-conditions {
       description
         "Data for general policy conditions, i.e., those
          not related to match-sets";

         leaf call-policy {
           type leafref {
             path "../../../../../../" +
               "rt-pol:policy-definitions/" +
               "rt-pol:policy-definition/rt-pol:name";
             require-instance true;
           }
           description
             "Applies the statements from the specified policy
              definition and then returns control the current
              policy statement. Note that the called policy may
              itself call other policies (subject to
              implementation limitations). This is intended to
              provide a policy 'subroutine' capability.  The
              called policy should contain an explicit or a
```

```
              default route disposition that returns an
              effective true (accept-route) or false
              (reject-route), otherwise the behavior may be
              ambiguous and implementation dependent";
         }

         leaf source-protocol {
           type identityref {
             base rt:control-plane-protocol;
           }
           description
             "Condition to check the protocol / method used to install
              the route into the local routing table";
         }
     }

     grouping policy-conditions-top {
       description
         "Top-level grouping for policy conditions";

       container conditions {
         description
           "Condition statements for the current policy statement";

         uses policy-conditions;

         uses generic-conditions;
       }
     }

     grouping policy-statements {
       description
         "Data for policy statements";

       leaf name {
         type string;
         description
           "Name of the policy statement";
       }
     }


     grouping policy-actions {
       description
         "Top-level grouping for policy actions";

       container actions {
         description
```

```
          "Top-level container for policy action statements";

        leaf policy-result {
          type policy-result-type;
          description
            "Select the final disposition for the route, either
             accept or reject.";
        }
        leaf set-metric {
          type uint32;
          description
            "Set a new metric for the route.";
        }
        leaf set-preference {
          type uint16;
          description
            "Set a new preference for the route.";
        }
      }
    }

    grouping policy-statements-top {
      description
        "Top-level grouping for the policy statements list";

      container statements {
        description
          "Enclosing container for policy statements";

        list statement {
          key "name";
          ordered-by user;
          description
            "Policy statements group conditions and actions
             within a policy definition.  They are evaluated in
             the order specified (see the description of policy
             evaluation at the top of this module.";

          uses policy-statements;

          uses policy-conditions-top;
          uses policy-actions;
        }
      }
    }


    grouping policy-definitions {
```

```
       description
         "This grouping provides policy definitions";

       leaf name {
         type string;
         description
           "Name of the top-level policy definition -- this name
           is used in references to the current policy";
       }
     }

     grouping apply-policy-import {
       description
         "Grouping for applying import policies";

       leaf-list import-policy {
         type leafref {
           path "/rt-pol:routing-policy/rt-pol:policy-definitions/" +
             "rt-pol:policy-definition/rt-pol:name";
           require-instance true;
         }
         ordered-by user;
         description
           "List of policy names in sequence to be applied on
            receiving a routing update in the current context, e.g.,
            for the current peer group, neighbor, address family,
            etc.";
       }

       leaf default-import-policy {
         type default-policy-type;
         default reject-route;
         description
           "Explicitly set a default policy if no policy definition
            in the import policy chain is satisfied.";
       }

     }

     grouping apply-policy-export {
       description
         "Grouping for applying export policies";

       leaf-list export-policy {
         type leafref {
           path "/rt-pol:routing-policy/rt-pol:policy-definitions/" +
             "rt-pol:policy-definition/rt-pol:name";
           require-instance true;
```

```
         }
         ordered-by user;
         description
           "List of policy names in sequence to be applied on
            sending a routing update in the current context, e.g.,
            for the current peer group, neighbor, address family,
            etc.";
       }

       leaf default-export-policy {
         type default-policy-type;
         default reject-route;
         description
           "Explicitly set a default policy if no policy definition
            in the export policy chain is satisfied.";
       }
     }


     grouping apply-policy {
       description
         "Configuration data for routing policies";

       uses apply-policy-import;
       uses apply-policy-export;

       container apply-policy-state {
         description
           "Operational state associated with routing policy";

         //TODO: identify additional state data beyond the intended
         //policy configuration.
       }

     }


     grouping apply-policy-group {
       description
         "Top level container for routing policy applications. This
          grouping is intended to be used in routing models where
          needed.";

       container apply-policy {
         description
           "Anchor point for routing policies in the model.
            Import and export policies are with respect to the local
            routing table, i.e., export (send) and import (receive),
```

```
            depending on the context.";

         uses apply-policy;

       }
     }

     container routing-policy {
       description
         "Top-level container for all routing policy";

       container defined-sets {
         description
           "Predefined sets of attributes used in policy match
            statements";

         uses prefix-set-top;
         uses neighbor-set-top;
         uses tag-set-top;
       }

       container policy-definitions {
         description
           "Enclosing container for the list of top-level policy
            definitions";

         list policy-definition {
           key "name";
           description
             "List of top-level policy definitions, keyed by unique
              name.  These policy definitions are expected to be
              referenced (by name) in policy chains specified in import
              or export configuration statements.";

           uses policy-definitions;

           uses policy-statements-top;
         }
       }
     }
   }
   <CODE ENDS>
```

11. Policy examples

Below we show an example of XML-encoded configuration data using the routing policy and BGP models to illustrate both how policies are defined, and also how they can be applied. Note that the XML has been simplified for readability.

```
<?yfile include="file:///tmp/routing-policy-example-draft.xml"?>
```

12. References

12.1. Normative references

[I-D.ietf-netmod-intf-ext-yang]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Common Interface Extension YANG Data Models", draft-ietf-netmod-intf-ext-yang-08 (work in progress), November 2019.

[I-D.ietf-netmod-sub-intf-vlan-model]
Wilton, R., Ball, D., tapsingh@cisco.com, t., and S. Sivaraj, "Sub-interface VLAN YANG Data Models", draft-ietf-netmod-sub-intf-vlan-model-06 (work in progress), November 2019.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8349] Lhotka, L., Lindem, A., and Y. Qu, "A YANG Data Model for Routing Management (NMDA Version)", RFC 8349, DOI 10.17487/RFC8349, March 2018, <https://www.rfc-editor.org/info/rfc8349>.

12.2. Informative references

[I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-07 (work in progress), October 2019.

Appendix A. Acknowledgements

The routing policy module defined in this draft is based on the OpenConfig route policy model. The authors would like to thank to OpenConfig for their contributions, especially Anees Shaikh, Rob Shakir, Kevin D'Souza, and Chris Chase.

The authors are grateful for valuable contributions to this document and the associated models from: Ebben Aires, Luyuan Fang, Josh George, Stephane Litkowski, Ina Minei, Carl Moberg, Eric Osborne, Steve Padgett, Juergen Schoenwaelder, Jim Uttaro, Russ White, and John Heasley.

Authors' Addresses

Yingzhen Qu
Futurewei
2330 Central Expressway
Santa Clara CA 95050
USA

Email: yingzhen.qu@futurewei.com

Jeff Tantsura
Apstra

Email: jefftant.ietf@gmail.com

Acee Lindem
Cisco
301 Mindenhall Way
Cary, NC 27513
US

Email: acee@cisco.com

Xufeng Liu
Volta Networks

Email: xufeng.liu.ietf@gmail.com