Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 30, 2016

A. Shaikh
Google
R. Shakir
Individual
K. D'Souza
C. Chase
AT&T

September 27, 2015

Routing Policy Configuration Model for Service Provider Networks
draft-ietf-rtgwg-policy-model-00

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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes a YANG [RFC6020] 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.

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. Model overview

The routing policy model is defined in two YANG modules, the main policy module, and an auxiliary module providing additional generic types. The model 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.

These modules make use of the standard Internet types, such as IP addresses, autonomous system numbers, etc., defined in RFC 6991 [RFC6991].

3. 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
                     ...
```

3.1. 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 across multiple routing protocols, and 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

- 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* [prefix-set-name]
      |     +--rw prefix-set-name    string
      |     +--rw prefix* [ip-prefix masklength-range]
      |        +--rw ip-prefix           inet:ip-prefix
      |        +--rw masklength-range    string
      +--rw neighbor-sets
      |  +--rw neighbor-set* [neighbor-set-name]
      |     +--rw neighbor-set-name    string
      |     +--rw neighbor* [address]
      |        +--rw address    inet:ip-address
      +--rw tag-sets
         +--rw tag-set* [tag-set-name]
            +--rw tag-set-name    string
            +--rw tag* [value]
               +--rw value    pt:tag-type
```

3.2. 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 statements
               +--rw statement* [name]
                  +--rw conditions
                     +--rw call-policy?
                     +--rw match-prefix-set!
                     |  +--rw prefix-set?
                     |  +--rw match-set-options?
                     +--rw match-neighbor-set!
                     |  +--rw neighbor-set?
                     |  +--rw match-set-options?
                     +--rw match-tag-set!
                     |  +--rw tag-set?
                     |  +--rw match-set-options?
                     +--rw install-protocol-eq?
                     +--rw igp-conditions
```

3.3. 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 (route-disposition)?
                     |  +--:(accept-route)
                     |  |  +--rw accept-route?   empty
                     |  +--:(reject-route)
                     |     +--rw reject-route?   empty
                     +--rw igp-actions
                        +--rw set-tag?   pt:tag-type
```

3.4. 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 (either explicit or by default), 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 4). 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.

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, with some major implementations only supporting a single subroutine, for example. 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.

4. Policy evaluation

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, 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 an alternate default action is specified for the chain).

5. 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 3) 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 config
|  |  +--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.

An example of using the apply-policy group in another routing model is shown below for BGP. Here, import and export policies are applied in the context of a particular BGP peer group. Note that the policy chains reference policy definitions by name that are defined in the routing policy model.

```
   +--rw bgp!
       +--rw peer-groups
          +--rw peer-group* [peer-group-name]
             +--rw peer-group-name
             +--rw config
             |  +--rw peer-as?
             |  +--rw local-as?
             |  +--rw peer-type?
             |  +--rw auth-password?
             |  +--rw remove-private-as?
             |  +--rw route-flap-damping?
             |  +--rw send-community?
             |  +--rw description?
             |  +--rw peer-group-name?
             +--ro state
             |  +--ro peer-as?
             |  +--ro local-as?
             |  +--ro peer-type?
             |  +--ro auth-password?
             |  +--ro remove-private-as?
             |  +--ro route-flap-damping?
             |  +--ro send-community?
             |  +--ro description?
             |  +--ro peer-group-name?
             |  +--ro total-paths?
             |  +--ro total-prefixes?
             +--rw apply-policy
             |  +--rw config
             |  |  +--rw import-policy*
             |  |  +--rw default-import-policy?
             |  |  +--rw export-policy*
             |  |  +--rw default-export-policy?
             |  +--ro state
             |     +--ro import-policy*
             |     +--ro default-import-policy?
             |     +--ro export-policy*
             |     +--ro default-export-policy?
             ...
```

6. 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 into the corresponding sections of the routing policy model.

```
+--rw routing-policy
   +--rw defined-sets
      +--rw prefix-sets
      |  +--rw prefix-set* [prefix-set-name]
      |     +--rw prefix-set-name
      |     +--rw prefix* [ip-prefix masklength-range]
      |        +--rw ip-prefix
      |        +--rw masklength-range
      +--rw neighbor-sets
      |  +--rw neighbor-set* [neighbor-set-name]
      |     +--rw neighbor-set-name
      |     +--rw neighbor* [address]
      |        +--rw address
      +--rw tag-sets
      |  +--rw tag-set* [tag-set-name]
      |     +--rw tag-set-name
      |     +--rw tag* [value]
      |        +--rw value
      +--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
         |     +--rw bgp-pol:community-member*
         +--rw bgp-pol:ext-community-sets
         |  +--rw bgp-pol:ext-community-set* [ext-community-set-name]
         |     +--rw bgp-pol:ext-community-set-name
         |     +--rw bgp-pol:ext-community-member*
         +--rw bgp-pol:as-path-sets
            +--rw bgp-pol:as-path-set* [as-path-set-name]
               +--rw bgp-pol:as-path-set-name
               +--rw bgp-pol:as-path-set-member*
```

7. 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.

8. 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].

9. YANG modules

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

9.1. Routing policy model

```
<CODE BEGINS> file routing-policy.yang
module routing-policy {

  yang-version "1";

  // namespace
  namespace "http://openconfig.net/yang/routing-policy";

  prefix "rpol";

  // import some basic types
  import ietf-inet-types { prefix inet; }
  import policy-types {prefix pt; }


  // meta
  organization
    "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.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 models (e.g., BGP) defined in other modules.

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 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 "2015-05-15" {
        description
          "Initial revision";
        reference "TBD";
      }


      // typedef statements

      typedef default-policy-type {
        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 default route disposition in
        a policy chain";
      }


      // grouping statements

      grouping generic-defined-sets {
        description
          "Data definitions for pre-defined sets of attributes used in
          policy match conditions.  These sets are generic and can
          be used in matching conditions in different routing
          protocols.";

        container prefix-sets {
          description
            "Enclosing container for defined prefix sets for matching";

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

            leaf prefix-set-name {
```

```
            type string;
            description
              "name / label of the prefix set -- this is used to
              reference the set in match conditions";
          }

          list prefix {
            key "ip-prefix masklength-range";
            description
              "List of prefix expressions that are part of the set";

            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-range {
              type string {
                pattern '^([0-9]+\.\.[0-9]+)|exact$';
              }
              description
                "Defines a range for the masklength, or 'exact' if
                the prefix has an exact length.

                Example: 10.3.192.0/21 through 10.3.192.0/24 would be
                expressed as prefix: 10.3.192.0/21,
                masklength-range: 21..24.

                Example: 10.3.192.0/21 would be expressed as
                prefix: 10.3.192.0/21,
                masklength-range: exact";
            }
          }
        }
      }

      container neighbor-sets {
        description
          "Enclosing container for defined neighbor sets for matching";

        list neighbor-set {
          key neighbor-set-name;
```

```
          description
              "Definitions for neighbor sets";

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

          list neighbor {
            key "address";
            description
                "list of addresses that are part of the neighbor set";

            leaf address {
              type inet:ip-address;
              description
                  "IP address of the neighbor set member";
            }
          }
        }
      }

      container tag-sets {
        description
          "Enclosing container for defined tag sets for matching";

        list tag-set {
          key tag-set-name;
          description
            "Definitions for tag sets";

          leaf tag-set-name {
            type string;
            description
              "name / label of the tag set -- this is used to reference
              the set in match conditions";
          }

          list tag {
            key "value";
            description
              "list of tags that are part of the tag set";

            leaf value {
              type pt:tag-type;
              description
```

```
                 "Value of the tag set member";
             }
           }
         }
       }
     }

     grouping local-generic-conditions {
       description
           "Condition statement definitions for consideration of a local
           characteristic of a route";

       leaf install-protocol-eq {
         type identityref {
           base pt:install-protocol-type;
         }
         description
           "Condition to check the protocol / method used to install
           which installed the route into the local routing table";
       }
     }

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

       leaf match-set-options {
         type pt:match-set-options-type;
         description
           "Optional parameter that governs the behaviour 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 pt:match-set-options-restricted-type;
         description
           "Optional parameter that governs the behaviour 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 generic-conditions {
       description "Condition statement definitions for checking
       membership in a generic defined set";

       container match-prefix-set {
         presence
           "The presence of this container indicates that the routes
           should match the prefix-set referenced.";

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

         leaf prefix-set {
           type leafref {
             path "/routing-policy/defined-sets/prefix-sets/" +
             "prefix-set/prefix-set-name";
             //TODO: require-instance should be added when it's
             //supported in YANG 1.1
             //require-instance true;
           }
           description "References a defined prefix set";
         }
         uses match-set-options-restricted-group;
       }

       container match-neighbor-set {
         presence
           "The presence of this container indicates that the routes
           should match the neighbour set referenced";

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

         leaf neighbor-set {
           type leafref {
             path "/routing-policy/defined-sets/neighbor-sets/" +
             "neighbor-set/neighbor-set-name";
             //TODO: require-instance should be added when it's
             //supported in YANG 1.1
             //require-instance true;
           }
           description "References a defined neighbor set";
         }
         uses match-set-options-restricted-group;
       }
```

```
       container match-tag-set {
         presence
           "The presence of this container indicates that the routes
           should match the tag-set referenced";

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

         leaf tag-set {
           type leafref {
             path "/routing-policy/defined-sets/tag-sets/tag-set" +
             "/tag-set-name";
             //TODO: require-instance should be added when it's
             //supported in YANG 1.1
             //require-instance true;
           }
           description "References a defined tag set";
         }
         uses match-set-options-restricted-group;
       }

       uses local-generic-conditions;
     }

     grouping igp-generic-conditions {
       description "grouping for IGP policy conditions";

     }


     grouping igp-conditions {
       description "grouping for IGP-specific policy conditions";

       container igp-conditions {
         description "Policy conditions for IGP attributes";

         uses igp-generic-conditions;

       }
     }

     grouping generic-actions {
       description
         "Definitions for common set of policy action statements that
         manage the disposition or control flow of the policy";

       choice route-disposition {
```

```
        description
          "Select the final disposition for the route, either
          accept or reject.";
        leaf accept-route {
          type empty;
          description "accepts the route into the routing table";
        }

        leaf reject-route {
          type empty;
          description "rejects the route";
        }
      }
    }


    grouping igp-actions {
      description "grouping for IGP-specific policy actions";

      container igp-actions {
        description "Actions to set IGP route attributes; these actions
        apply to multiple IGPs";

        leaf set-tag {
          type pt:tag-type;
          description
            "Set the tag value for OSPF or IS-IS routes.";
        }
      }
    }

    container routing-policy {
      description
        "top-level container for all routing policy configuration";

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

        uses generic-defined-sets;
        // uses bgp-defined-sets;
        // don't see a need for IGP-specific defined sets at this point
        // e.g., for OSPF, IS-IS, etc.
      }

      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/
            export configuration statements.";


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

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

            list statement {
              key name;
              // TODO: names of policy statements within a policy defn
              // should be optional, however, YANG requires a unique id
              // for lists; not sure that a compound key works either;
              // need to investigate further.
              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.";

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

              container conditions {

                description "Condition statements for this
                  policy statement";

                leaf call-policy {
```

```
                 type leafref {
                   path "/rpol:routing-policy/" +
                     "rpol:policy-definitions/" +
                     "rpol:policy-definition/rpol:name";
                   //TODO: require-instance should be added when it's
                   //supported in YANG 1.1
                   //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";
               }
               uses generic-conditions;
               uses igp-conditions;
             }

             container actions {

               description "Action statements for this policy
                     statement";

               uses generic-actions;
               uses igp-actions;
             }
           }
         }
       }
     }
   }

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

     leaf-list import-policy {
       type leafref {
         path "/rpol:routing-policy/rpol:policy-definitions/" +
           "rpol:policy-definition/rpol:name";
         //TODO: require-instance should be added when it's
```

```
          //supported in YANG 1.1
          //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.";
      }

      leaf-list export-policy {
        type leafref {
          path "/rpol:routing-policy/rpol:policy-definitions/" +
            "rpol:policy-definition/rpol:name";
          //TODO: require-instance should be added when it's
          //supported in YANG 1.1
          //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-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.";

         container config {
           description
             "Policy configuration data.";

           uses apply-policy-config;
         }

         container state {

           config false;
           description
             "Operational state for routing policy";

           uses apply-policy-config;
           uses apply-policy-state;
         }
       }
     }
   }
   <CODE ENDS>
```

9.2. Routing policy types

```
   <CODE BEGINS> file policy-types.yang
   module policy-types {

     yang-version "1";

     // namespace
     namespace "http://openconfig.net/yang/policy-types";
```

```
  prefix "ptypes";

  // import some basic types
  import ietf-yang-types { prefix yang; }


  // meta
  organization
    "OpenConfig working group";

  contact
    "OpenConfig working group
    netopenconfig@googlegroups.com";

  description
    "This module contains general data definitions for use in routing
    policy.  It can be imported by modules that contain protocol-
    specific policy conditions and actions.";

  revision "2015-05-15" {
    description
      "Initial revision";
    reference "TBD";
  }

  // identity statements

  identity attribute-comparison {
    description
      "base type for supported comparison operators on route
      attributes";
  }

  identity attribute-eq {
    base attribute-comparison;
    description "== comparison";
  }

  identity attribute-ge {
    base attribute-comparison;
    description ">= comparison";
  }

  identity attribute-le {
    base attribute-comparison;
    description "<= comparison";
  }
```

```
     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";
     }

     typedef match-set-options-restricted-type {
       type enumeration {
         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";
         }
       }
       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.  Note this type is a
         restricted version of the match-set-options-type.";
         //TODO: restriction on enumerated types is only allowed in
         //YANG 1.1.  Until then, we will require this additional type
     }

     grouping attribute-compare-operators {
       description "common definitions for comparison operations in
       condition statements";

       leaf operator {
```

```
          type identityref {
            base attribute-comparison;
          }
          description
            "type of comparison to be performed";
        }

      leaf value {
        type uint32;
        description
          "value to compare with the community count";
      }
    }

    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
      hexidecimal integer";
      reference
        "RFC 2178 OSPF Version 2
        RFC 5130 A Policy Control Mechanism in IS-IS Using
        Administrative Tags";
    }

    identity install-protocol-type {
      description
        "Base type for protocols which can install prefixes into the
        RIB";
    }

    identity BGP {
      base install-protocol-type;
      description "BGP";
      reference "RFC 4271";
    }

    identity ISIS {
      base install-protocol-type;
      description "IS-IS";
      reference "ISO/IEC 10589";
    }

    identity OSPF {
      base install-protocol-type;
```

```
      description "OSPFv2";
      reference "RFC 2328";
    }

    identity OSPF3 {
      base install-protocol-type;
      description "OSPFv3";
      reference "RFC 5340";
    }

    identity STATIC {
      base install-protocol-type;
      description "Locally-installed static route";
    }

    identity DIRECTLY-CONNECTED {
      base install-protocol-type;
      description "A directly connected route";
    }

    identity LOCAL-AGGREGATE {
      base install-protocol-type;
      description "Locally defined aggregate route";
    }
  }
  <CODE ENDS>
```

10. 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.

```
<routing-policy>

  <defined-sets>
    <prefix-set name="prefix-set-A">
      <prefix>
        <address>A1</address>
        <masklength>M1</masklength>
      </prefix>
      <prefix>
        <address>A2</address>
        <masklength>M2</masklength>
      </prefix>
```

```
         <prefix>
           <address>A3</address>
           <masklength>M3</masklength>
         </prefix>
       </prefix-set>

       <tag-set>
         <tag-set-name>cust-tag1</tag-set-name>
         <tag value="10" />
       </tag-set>

       <community-set name="community-set-A">
         <community-member>C1</community-member>
         <community-member>C2</community-member>
         <community-member>C3</community-member>
       </community-set>
       <community-set name="community-set-B">
         <community-member>C5</community-member>
         <community-member>C6</community-member>
         <community-member>C7</community-member>
       </community-set>

       <as-path-set name="as-path-set-A">
         <as-path-set-member>AS1</as-path-set-member>
         <as-path-set-member>AS2</as-path-set-member>
         <as-path-set-member>ASx</as-path-set-member>
       </as-path-set>

     </defined-sets>

     <!-- policy 1:
       if community in community-set-A then local-pref = 10
       if origin = IGP then accept route
     -->
     <policy-defintion name="policy 1">
       <policy-statements>
         <statement name="depref-community-A">
           <conditions>
             <match-community-set>
               <community-set>community-set-A</community-set>
             </match-community-set>
           </conditions>
           <actions>
             <set-local-pref>10</set-local-pref>
           </actions>
         </statement>
         <statement name="accept-igp">
           <conditions>
```

```
            <origin-eq>IGP</origin-eq>
          </conditions>
          <actions>
            <accept-route />
          </actions>
        </statement>
      </policy-statements>
    </policy-defintion>

    <!-- policy 2:
      if community matches-exactly community-set-B and AS
      path in as-path-set-A then reject
    -->

    <policy-defintion name="policy 2">
      <statement name="drop-community-B-aspath-A">
        <conditions>
          <match-community-set>
            <community-set>community-set-B</community-set>
            <match-set-options>ALL</match-set-options>
          </match-community-set>
          <match-as-set>
            <as-set>as-path-set-A</as-set>
          </match-as-set>
        </conditions>
        <actions>
          <reject-route />
        </actions>
      </statement>
    </policy-defintion>

   <!-- policy 3:
      if community matches-exactly community-set-A
        then accept
    -->

    <policy-definition name="policy 3">
      <statement name="accept-community-A">
        <conditions>
          <match-community-set>
            <community-set>community-set-A</community-set>
            <match-set-options>ALL</match-set-options>
          </match-prefix-set>
        </conditions>
        <actions>
          <accept-route />
        </actions>
      </statement>
```

```
     </policy-definition>

    <!-- policy export-tagged-BGP:
       if route from OSPFv3 and tag=cust-tag1
         then accept
     -->

     <policy-definition name="export-tagged-BGP">
       <statement>
         <conditions>
           <install-protocol-eq>OSPFV3</install-protocol-eq>
           <match-tag-set>cust-tag1</match-tag-set>
         </conditions>
         <actions>
           <accept-route />
         </actions>
       </statement>
     </policy-definition>


   </routing-policy>

   <!-- import policy chain for BGP neighbor -->
   <bgp>
     <neighbor>
       <neighbor-address>172.95.25.2</neighbor-address>
       <peer-AS>ASY</peer-AS>
       <description>regional peer ASY</description>
       <peer-type>EXTERNAL</peer-type>
       <advertise-inactive-routes>true</advertise-inactive-routes>
       <use-multiple-paths>
         <ebgp>
           <maximum-paths>4</maximum-paths>
         </ebgp>
       </use-multiple-paths>
       <import-policies>
         <policyref>policy 2</policyref>
         <policyref>policy 3</policyref>
         <default-policy>REJECT-ROUTE</default-policy>
       </import-policies>
     </neighbor>
   </bgp>
```

11. References

11.1. Normative references

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2014.

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC3688] Mealling, M., "The IETF XML Registry", RFC 3688, January 2004.

11.2. Informative references

[I-D.ietf-idr-bgp-model] Shaikh, A., Shakir, R., Patel, K., Hares, S., D'Souza, K., Bansal, D., Clemm, A., Alex, A., Jethanandani, M., and X. Liu, "BGP Model for Service Provider Networks", draft-ietf-idr-bgp-model-00 (work in progress), July 2015.

[I-D.openconfig-netmod-opstate] Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-00 (work in progress), March 2015.

Appendix A. Acknowledgements

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

Appendix B. Change summary

B.1. Changes between revisions draft-shaikh-rtgwg-policy-model and -00

This revision updates the draft name to reflect adoption as a working document in the RTGWG. Minor changes include updates to references and updated author contact information.

Authors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: aashaikh@google.com

Rob Shakir
Individual

Email: rjs@rob.sh

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
US

Email: kd6913@att.com

Chris Chase
AT&T
9505 Arboretum Blvd
Austin, TX
US

Email: chase@labs.att.com