Routing Policy Configuration Model for Service Provider Networks
draft-shaikh-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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 27, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:
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-definition* [name]
        +--rw name         string
        +--rw statement* [name]
            +--rw name         string
            +--rw conditions!
            |     ...
            +--rw actions!
            ...  
```

3.1. 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:
o ALL - match is true only if the given value matches all members of the set.

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

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

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-definition* [name]
      +--rw statement* [name]
         +--rw conditions!
            +--rw call-policy?   -> /routing-policy/...
            +--rw match-prefix-set? -> /routing-policy/...
            +--rw match-neighbor-set? -> /routing-policy/...
            +--rw match-tag-set?   -> /routing-policy/...
            +--rw install-protocol-eq? identityref
            +--rw igp-conditions
```

3.2. 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.
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 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.

```plaintext
  +--rw apply-policy
    |  +--rw import-policies*  -> /rpol:routing-policy/...
    |  +--rw default-import-policy?  default-policy-type
    |  +--rw export-policies*  -> /rpol:routing-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.

```plaintext
  +--rw peer-group* [group-name]
    |  |  +--rw group-name                string
    |  |  +--ro bgp-group-common-state
    |  |  +--rw description?              string
    |  |  +--rw graceful-restart!
    |  |  |  +--rw restart-time?        uint16
    |  |  |  +--rw stale-routes-time?   decimal64
    |  |  +--rw apply-policy
    |  |  |  +--rw import-policies*  -> /rpol:routing-policy/...
    |  |  |  +--rw default-import-policy?  default-policy-type
    |  |  |  +--rw export-policies*  -> /rpol:routing-policy/...
    |  |  |  +--rw default-export-policy?  default-policy-type
    |  ...
```
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 [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.

```plaintext
+--rw routing-policy
   +--rw defined-sets!
      |  +--rw prefix-set* [prefix-set-name]
      |     +--rw prefix-set-name    string
      |     +--rw prefix* [address masklength masklength-range]
      |        +--rw address             inet:ip-address
      |        +--rw masklength          uint8
      |        +--rw masklength-range    string
      |  +--rw neighbor-set* [neighbor-set-name]
      |     +--rw neighbor-set-name    string
      |     +--rw neighbor* [address]
      |        +--rw address    inet:ip-address
      |  +--rw tag-set* [tag-set-name]
      |     +--rw tag-set-name    string
      |     +--rw tag* [value]
      |        +--rw value    pt:tag-type
   +--rw bgp-pol:bgp-defined-sets
      +--rw bgp-pol:community-set* [community-set-name]
      |     +--rw bgp-pol:community-set-name    string
      |     +--rw bgp-pol:community-members*    union
      +--rw bgp-pol:ext-community-set* [ext-community-set-name]
      |     +--rw bgp-pol:ext-community-set-name    string
      |     +--rw bgp-pol:ext-community-members*    union
      +--rw bgp-pol:as-path-set* [as-path-set-name]
      |     +--rw bgp-pol:as-path-set-name    string
      |     +--rw bgp-pol:as-path-set-members*    string
```

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";
</CODE>
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 multiple 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 "2014-11-30" {
  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";
}

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";
}

// 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.";

description
  "Definitions for 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 "address masklength masklength-range";
description
"list of prefix expressions that are part of the set";

leaf address {
  type inet:ip-address;
  mandatory true;
  description
  "address portion of the prefix";
}

leaf masklength {
  type uint8 {
    // simple range covers both ipv4 and ipv6 --
    // could separate this into different types
    // for IPv4 and IPv6 prefixes
    range 0..128;
  }
  mandatory true;
  description
  "masklength for the prefix specification";
}

leaf masklength-range {
  type string {
    // pattern modeled after ietf-inet-types
    pattern '\(([0-9])\|([1-9]\([0-9]\))\|([1-9][0-9]-[0-9]\))\|' + '\(([2-4])\|([0-9]-[0-9]\))\|' + '\(([0-9])\|([1-9]\([0-9]\))\|([0-9]-[0-9]\))\|' + '\(([2-4])\|([0-9]-[0-9]\))\|' + '\(([0-9])\|([1-9]\([0-9]\))\|([1-9][0-9]-[0-9]\))\|' + '\(([2-4])\|([0-9]-[0-9]\))\|';
  }
  description
  "Defines an optional range for the masklength. Absence
  of the masklength-length implies that the prefix has an
  exact masklength given by the masklength parameter.
  Example: 10.3.192.0/21 through 10.3.192.0/24 would be
  expressed as address: 10.3.192.0, masklength: 21,
  masklength-range: 21..24";
}

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";
  }
}

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 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 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-set" + ""/prefix-set-name"; require-instance true;
            }
            description "References a defined prefix set";
        }
        uses match-set-options-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-set" + "/neighbor-set-name";
    require-instance true;
  }
  description "References a defined neighbor set";
}
uses match-set-options-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-set" + "/tag-set-name";
      require-instance true;
    }
    description "References a defined tag set";
  }
  uses match-set-options-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";
    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 {
            presence "Container for sets defined for matching in policy
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.
}

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";
  }
}

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 {
presence "conditions";
description "Condition statements for this policy statement";

leaf call-policy {
  type leafref {
    path "/rpol:routing-policy/rpol:policy-definition/rpol: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";
}
uses generic-conditions;
uses igp-conditions;
}

container actions {
  presence "actions";
description "Action statements for this policy statement";

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

grouping apply-policy-group {

description "configuration for applying policies";

container apply-policy {

description "Anchor point for routing policies in the configuration. Import and export policies are with respect to the local routing table, i.e., export (send) and import (receive), depending on the context."
}
leaf-list import-policies {
  type leafref {
    path "/rpol:routing-policy/rpol:policy-definition" + "/*/rpol: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.";
}
leaf-list export-policies {
  type leafref {
    path "/rpol:routing-policy/rpol:policy-definition" + "/*/rpol: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.";
}
9.2. Routing 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-inet-types { prefix inet; }
  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 "2014-11-30" {
    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";
  }
```

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 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";
    }
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.

```xml
<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-set>
  </defined-sets>
</routing-policy>
```
<defset>
  
  <commset name="community-set-A">
    <commmember>C1</commmember>
    <commmember>C2</commmember>
    <commmember>C3</commmember>
  </commset>

  <commset name="community-set-B">
    <commmember>C5</commmember>
    <commmember>C6</commmember>
    <commmember>C7</commmember>
  </commset>

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

  <policydefinition name="policy 1">
    <policystatements>
      <statement name="depref-community-A">
        <conditions>
          <matchcommunityset>
            <communityset>community-set-A</communityset>
          </matchcommunityset>
        </conditions>
        <actions>
          <setlocalpref>10</setlocalpref>
        </actions>
      </statement>
      <statement name="accept-igp">
        <conditions>
          <origineq>IGP</origineq>
        </conditions>
        <actions>
          <acceptroute />
        </actions>
      </statement>
    </policystatements>
  </policydefinition>
</defset>
<!-- policy 2:
    if community matches-exactly community-set-B and AS path in as-path-set-A
    then reject
-->
<policy-definition 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-definition>

<!-- 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


11.2. Informative references


Appendix A. Acknowledgements

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

Authors’ Addresses

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

Email: aashaikh@google.com

Rob Shakir
BT
pp. C3L, BT Centre
81, Newgate Street
London EC1A 7AJ
UK

Email: rob.shakir@bt.com
URI: http://www.bt.com/